universität wien

D I S S E R T A T I O N

Titel der Dissertation

Definition and Enforcement of Access Constraints in Collaborative Processes

verfasst von

Mag. Patrick Gaubatz

angestrebter akademischer Grad

Doktor der Wirtschaftswissenschaften (Dr.rer.oec.)

Wien, 2015

Studienkennzahl lt. Studienblatt:	A 784 175
Dissertationsgebiet lt. Studienblatt:	Wirtschaftsinformatik
Betreuer:	Univ.-Prof. Dr. Uwe Zdun

Declaration of Authorship

I, Patrick Gaubatz, declare that this thesis with the title, "Enforcing Access Constraints in Collaborative Processes" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Abstract

A collaborative process is a structured or unstructured process where two or more different stakeholders are working together to fulfill a shared, collective and bounded goal. Especially in today's technology- and information-driven society, participation in such collaborative processes is common place. Information technology and the automation provided by IT systems lead to an ever-increasing shift towards virtual collaborative processes. In general, collaborative processes are often subject to security policies, business rules, laws and regulations. For instance, a legal contract has to be signed by all contractual partners and certified by a notary. In virtual collaborative processes, such restrictions could be enforced using different types of access constraints. In particular, they could strictly constrain the way each stakeholder participates in a collaborative process. However, developing and maintaining means for defining and enforcing different kinds of access constraints and access control in general is a cross-cutting concern that is inherently prone to increase the complexity of software systems and to degrade the software system's code quality. The diverse variety of collaborative process types, which ranges from structured and rigid business processes to unstructured and flexible real-time collaborative Web applications (such as collaborative text editing), to be made subject to access constraints, requires custom, domain-specific enforcement mechanisms and approaches, which would significantly increase the overall development effort. Particularly, real-time collaborative Web applications require these enforcement mechanisms to be scalable and computationally efficient too.

In this thesis, we propose a model-driven approach for defining and enforcing different types of access constraints in different types of collaborative processes. Our approach aims for tackling the aforementioned challenges while, at the same time, reducing the required development and maintenance effort. Based on this approach, we explore the implications of changing access constraints dynamically at runtime and devise a novel set of generic and reusable consistency checking and conflict resolution strategies. In the context of real-time collaborative Web applications, we introduce and evaluate the concept of dynamic view customization as a means for enforcing access constraints. In addition, we investigate offline-editing scenarios in the context of access constrained, real-time collaborative Web documents. In particular, we demonstrate how a combination of client-side access control enforcement and a corresponding document merging approach can be used to enable mobile workers to participate in a real-time collaborative process while their computing devices are temporarily not connected to any network. Our proposed model-driven approach as well as the accompanying concepts have been implemented in various research prototypes, thereby showing the feasibility and effectiveness. Extensive performance measurements demonstrate that our concepts and prototypes are applicable in real-life collaborative processes as well as in a larger Web context. Controlled experiments provide evidence that automatic access constraint enforcement increases both the effectivity and efficiency of users. In summary, this thesis has gone a considerable step towards enhancing our understanding of making collaborative processes subject to diverse kinds of access constraints.

Zusammenfassung

Ein kollaborativer Prozess ist ein strukturierter oder unstrukturierter Prozess in dem zwei oder mehrere Akteure zusammenarbeiten, um ein gemeinsames, abgestecktes Ziel zu erreichen. Vor allem in der heutigen Technologie- und Informations-gestützten Gesellschaft ist die Teilnahme an solchen kollaborativen Prozessen alltäglich. Die wohlbekannten Vorteile von Informationstechnologien und Automatisierung durch IT-Systeme hat zu einer stetig zunehmenden Virtualisierung von kollaborativen Prozessen geführt. Kollaborative Prozesse werden oft Sicherheitsrichtlinien, Unternehmensrichtlinien, Gesetzen und Regulierungen unterworfen. So muss beispielsweise ein Vertrag von allen Vertragspartnern unterschrieben und von einem Notar beglaubigt werden. In virtuellen kollaborativen Prozessen könnten diese Einschränkungen mittels Zugriffs-Constraints durchgesetzt werden, welche die Art und Weise wie jeder Akteur am kollaborativen Prozess teilnimmt einschränken. Die Entwicklung und Pflege von Mechanismen zur Definition und Durchsetzung von verschiedensten Zugriffs-Constraints und Zugriffskontrolle im Allgemeinen, ist ein Cross-Cutting-Concern und daher besonders anfällig, die Komplexität von Software-Systemen zu erhöhen und die Code-Qualität des Software-Systems zu verschlechtern. Die Vielfalt an verschiedenartigen Typen von kollaborativen Prozessen, welche von strukturierten und starren Business-Prozessen bis hin zu flexiblen Echtzeit-Kollaborations-Web-Applikationen (z.B. kollaboratives Editieren von Texten) reicht, welche Zugriffs-Constraints unterworfen werden sollen, benötigen maßgeschneiderte, Domänen-spezifische Durchsetzungsmechanismen, welche den Gesamtentwicklungsaufwand nochmals erhöhen. Insbesondere Echtzeit-Kollaborations-Web-Applikationen verlangen nach skalierbaren und effizienten Durchsetzungsmechanismen.

In dieser Dissertation stellen wir einen Modell-getriebenen Ansatz zur Definition und Durchsetzung von Zugriffs-Constraints für verschiedenste Arten von kollaborativen Prozessen. Unser Ansatz zielt darauf ab, die zuvor genannten Herausforderungen zu bewältigen und den Entwicklungs- und Wartungsaufwand zu reduzieren. Auf Basis dieses Ansatzes untersuchen wir die Auswirkungen von Zugriffs-Constraint-Änderungen zur Laufzeit und beschreiben einen generischen Mechanismus zur Konsistenzprüfung und Konfliktauflösung. Im Kontext von Echtzeit-Kollaborations-Web-Applikationen beschreiben und evaluieren wir das Konzept der dynamischen View-Konfiguration zur Durchsetzung von Zugriffs-Constraints. Außerdem erforschen wir Offline-Bearbeitungsszenarios im Kontext von Zugriffs-Constrains und Echtzeit-Kollaborations-Web-Dokumenten. Dabei demonstrieren wir, wie eine Kombination von Client-seitiger Zugriffskontrolle und ein Dokumenten-Merge-Mechanismus mobilen Arbeitern ermöglicht, an einem Echtzeit-kollaborativen Prozess teilzunehmen, während deren Geräte temporär nicht mit einem Netzwerk verbunden sind. Wir belegen die Machbarkeit und Effektivität unseres Modell-getriebenen Ansatzes und dessen zugehörigen Konzepten mit Hilfe von Forschungsprototypen. Umfangreiche Performance-Messungen zeigen, dass unsere Konzepte und Prototypen auch in der Praxis bzw. einem größeren Web-Kontext anwendbar sind. Kontrollierte Experimente weisen nach, dass das automatische Durchsetzen von Zugriffs-Constraints sowohl die Effektivität als auch die Effizienz von Benutzern steigert. Diese Dissertation erweitert unser Verständnis von der Unterwerfung von kollaborativen Prozessen durch verschiedenste Arten von Zugriffs-Constraints.

Acknowledgments

Finishing this thesis makes me immensely proud and I can, without any hesitation, admit that the last four years have been the most challenging and demanding, but also the most rewarding years of my life, so far. A work of this scale is never undertaken alone and I am deeply indebted to those that supported, influenced, inspired, motivated and enabled me to pursue this momentous and memorable journey.

First of all, I would like to express my sincere thanks to my supervisor, Prof. Uwe Zdun, who has been an inexhaustible source of inspiration and motivation, for guiding me throughout my studies and for always being available when I felt I was lost. I am also most grateful to Prof. Cesare Pautasso for serving as examiner of my thesis. Special thanks go to my colleagues, Mark Strembeck, Waldemar Hummer, Ioanna Lytra, Thomas Quirchmayr and Stefan Sobernig for the endless, but enjoyable and fruitful discussions and collaboration eventually leading to numerous successful paper projects. I would like to thank the whole Software Architecture Group, most notably Thomas Haitzer, Ioanna Lytra, Gerhard Pernecker, Simon Tragatschnig and Huy Tran, for providing such a cordial, productive and pleasant working environment.

Finally, I am sincerely thankful to family and friends, who always cheered me up and also provided the occasional but definitely necessary distractions from work. Most of all, I wanted to thank the most important person in my life, my girlfriend Claudia, who had to suffer with me through all the ups and downs and who was always there for me. Without her, I could not have finished this thesis. Thank you!

Patrick Gaubatz

Hainburg an der Donau, December 2014

Contents

Publications

This cumulative doctoral dissertation consists of work that has either been published in scientific conferences, workshops, journals and books or is currently under review. The following list of publications is included in this thesis:

- P. Gaubatz, I. Lytra, and U. Zdun. Automatic Enforcement of Constraints in Real-time Collaborative Architectural Decision Making. *Journal of Systems and Software*, accepted for publication in January 2015
- I. Lytra, P. Gaubatz, and U. Zdun. Two Controlled Experiments on Model-based Architectural Decision Making. submitted to *Information and Software Technology*, submitted first revision in January 2015
- T. Quirchmayr, P. Gaubatz, M. Strembeck, and U. Zdun. Consistency Checking and Resolution Strategies for Runtime Conflicts Resulting from Changes in Process-related RBAC Models. submitted to *Advances in Verifiably Secure Process-aware Information Systems*, submitted in June 2014
- P. Gaubatz, W. Hummer, U. Zdun, and M. Strembeck. Enforcing Entailment Constraints in Offline Editing Scenarios for Real-time Collaborative Web Documents. In *29th Symposium On Applied Computing*, Gyeongju, Korea, March 2014
- W. Hummer, P. Gaubatz, M. Strembeck, U. Zdun, and S. Dustdar. Enforcement of Entailment Constraints in Distributed Service-based Business Processes. *Information and Software Technology*, 55(11), November 2013
- P. Gaubatz, W. Hummer, U. Zdun, and M. Strembeck. Supporting Customized Views for Enforcing Access Control Constraints in Real-time Collaborative Web Applications. In *13th International Conference on Web Engineering*, Aalborg, Denmark, July 2013
- P. Gaubatz and U. Zdun. Supporting Entailment Constraints in the Context of Collaborative Web Applications. In *28th Symposium On Applied Computing*, Coimbra, Portugal, March 2013
- P. Gaubatz and U. Zdun. UML2 Profile and Model-Driven Approach for Supporting System Integration and Adaptation of Web Data Mashups. In *4th International Workshop on Lightweight Integration on the Web*, Berlin, Germany, July 2012
- W. Hummer, P. Gaubatz, M. Strembeck, U. Zdun, and S. Dustdar. An Integrated Approach for Identity and Access Management in a SOA Context. In *16th ACM Symposium on Access Control Models and Technologies*, Innsbruck, Austria, June 2011

Chapter 1

Introduction

Automation provided by IT systems is a continuing trend that fundamentally changes *what*, *how*, *when* and *where* we work and conduct business. Especially repetitive work tasks are being standardized, automated, monitored and streamlined, in order to increase productivity and to eliminate human error as much as possible. *Office automation* aims for the automation of creating, collecting, modifying, archiving and relaying of office data. The last decades were and future decades will be characterized by the virtualization and entailing automation of more and more of such office related work tasks. Nowadays, the vast majority of office data that is being created, processed and relayed is electronic data. Striving for the frequently mentioned *paperless office*, printed documents have been replaced by electronic forms or documents, file cabinets have been replaced by databases and document storages, and conventional postal delivery services ("snail mail") is literally facing extinction because of the e-mail. Despite this radical and ongoing shift towards the virtualization of office work, there will (at least for the foreseeable future) always remain situations, work tasks or processes that require some kind of human-to-human interaction. Where formerly sheets of paper where physically handed from one person to the next, in order to complete a record that requires to be handled by more than one person, such collaborative processes are increasingly transformed into virtual collaborative processes.

Motivated by these trends and observations, this PhD thesis ultimately aims for improving the way different persons collaborate within the boundaries of virtual collaborative process. In the context of this thesis, we define a collaborative process as follows:

Definition 1 *A* COLLABORATIVE PROCESS *is a structured or unstructured process where two or more different stakeholders are working together to fulfill a shared, collective and bounded goal.*

An exemplary collaborative process is the process of putting a legal contract in place. It involves at least two different stakeholders (i.e., contractual partners), that have to negotiate the contract's exact terms, in order to reach their shared, collective and bounded

goal, i.e., a legally binding contract. The aforementioned technological advances lead to an ever-increasing shift towards virtual collaborative processes, such as collaborative document editing, decision making, (software) project management/development, white-boarding, groupware, wikis as well as a plethora of so-called *process aware information systems* (PAIS), i.e., *software that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models* [18].

Specifying *who* is supposed to do *what* within a collaborative process, is essential for successful collaboration. Also, collaborative processes are often subject to security policies, business rules, laws and regulations, such as the Basel II/III Accords, the International Financial Reporting Standards (IFRS), the Health Insurance Portability and Accountability Act (HIPPA), or the Sarbanes-Oxley Act (SOX). For instance, for a valid, legally binding contract, it is required, that it gets signed by all contractual partners and certified by a notary. Or, in the e-health context there could be the requirement, that for every report that is filed by a doctor, a second doctor needs to approve and sign the report to realize the well-known four eyes principle. In virtual collaborative processes, such restrictions can be enforced using different types of access constraints. We define an access constraint as follows:

Definition 2 *An* ACCESS CONSTRAINT *restricts or mandates the way each stakeholder must or must not participate in a collaborative process.*

In other words, access constraints provide means for precisely defining *who* must or must not do *what* within a collaborative process. Thus, they are crucial means for preventing chaos, dysfunctional ambiguities or conflicts within collaborative processes and satisfying complex requirements that stem from laws, regulations or any other compliance rules.

However, from a software developer's point of view, developing and maintaining means for defining and enforcing different kinds of access constraints is a complex, error-prone and time-consuming endeavor. Consequently, this thesis strives for enhancing our understanding of making collaborative processes subject to diverse kinds of access constraints and finding ways of reducing the effort, that is required to implement and maintain access constrained collaborative processes.

The remainder of this cumulative PhD thesis is structured as follows. Chapter 2 defines relevant terms and provides a literature overview of topics that build the foundation this thesis. Chapter 3 details the research problem domain and context, highlights research challenges and states the research questions. Chapter 4 discusses the pursued research method, lists all scientific papers that have been published and details the scientific contributions that have been made in the course of this thesis. Finally, Chapter 5 concludes this thesis and gives an outlook on potential future work.

Chapter 2

Background

This chapter provides a literature overview of the relevant terms and topics and builds the foundation of this thesis.

2.1 Business Processes and Workflows

In essence, a *Business Process* is an ordered set of activities or tasks that produce a certain desirable result [105]. The concept of business processes emerged from earlier work in the context of office information systems (see, e.g., [20, 46, 104]), seeking for ways to improve the efficiency of organizations. *Business Process Management* (BPM) can be defined as "supporting business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information" [96].

A cornerstone of BPM's promise to improve operational efficiency is automation of processes. In order to automate the executing of processes, business processes need to be specified using a *process modeling language*. A process modeling language provides means for precisely defining the task as well as their execution order within a business process. Several modeling languages and approaches have been introduced over the years. Popular examples include Petri nets [64], the Unified Modeling Language (UML) [67], the Business Process Modeling And Notation (BPMN) [68], Yet Another Workflow Language (YAWL) [95], or the Web Services Business Process Execution Language (WS-BPEL) [66]. A central idea in this context is the distinction between *process type* (e.g., "Patient admission") and *process instance* (e.g., "Admission of patient Patrick Gaubatz"). Thereby, process types are defined using a particular process modeling language and each process type can have an arbitrary number of instances, which are handled individually. Note that business processes can be modeled at different abstraction levels. To this end, a business process that is in its execution level is often referred to as a *workflow* [45, 51].

Workflows can be executed within the context of workflow management systems. In

recent years, such systems have evolved into *Process-aware Information Systems* (PAIS). A PAIS can be defined as "a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models" [18, 94]. According to this definition, many other types of information systems can be considered to be "process aware" even if their processes are hard-coded or only used implicitly [93]. Popular example of PAIS are, "classical" workflow management (WFMS) systems, enterprise resource planning (ERP) systems, case handling systems, product data management (PDM) systems, customer relationship management (CRM) systems, or hospital information systems. Following the aforementioned definition of PAIS, we argue that real-time collaborative Web applications, which will be discussed in the next section, may also be classified as PAIS in the broader sense. That is, real-time collaborative Web applications typically have an implicit underlying process, such as "a particular form or document needs to be filled out completely". Contrary to business processes, in such real-time collaborative processes the precise set and sequence of tasks that need to be performed in order to complete the process is not defined or predetermined.

2.2 Real-time Collaborative Web Applications

Real-time collaborative Web applications are Web applications that employ *synchronous distributed interaction* [22], a collaboration form in the context of real-time groupware, that allows multiple users to edit a shared artifact concurrently, at the same time. *Groupware* is a "computer-based system that supports groups of people engaged in a common task (or goal) and that provides an interface to a shared environment" [22]. Groupware can be distinguished into *real-time groupware* and *non-real-time groupware*. While the former requires that users are actively (and synchronously) using the application at the same time, the latter describes an asynchronous style of communication and collaboration that is employed, for instance, in *asynchronous conferencing* software, such as online forums, blogs and wikis (see, e.g., [61, 72]) The groundwork of real-time groupware has been laid by Douglas Engelbart's "The Mother of All Demos" [24], demonstrating the first multi-user text editor in 1968. Nearly three decades later, Web browsers are becoming the common platform of choice for delivering real-time groupware applications (see, e.g., [43, 63, 99]). Popular examples of Web-based real-time groupware range from text editing (see, e.g., Google Docs[1] or Sharelatex[2]), to Integrated Development Environments (see, e.g., Cloud9[3] or Collabode [40]), to modeling tools (see, e.g., Creatly[4] or Cacoo[5]).

[1] http://docs.google.com
[2] http://sharelatex.com
[3] http://c9.io
[4] http://creately.com
[5] http://cacoo.com

The cornerstone of real-time groupware is *concurrency control*. The purpose of concurrency control is to ensure data consistency and resolving conflicts between users' simultaneous operations [21]. In both real-time and non-real-time groupware, conflicts are inherent and happen when two (or more) participants change the very same part of a shared document. An exemplary conflict might be a situation where one user fixes a typo within a particular word, while another user removes the very same word, at the same time. Concurrency control approaches can be distinguished in *optimistic concurrency control* and *pessimistic concurrency control* approaches. The prevailing optimistic approaches (see, e.g, *Operational Transformation* [21] and *Differential Synchronization* [31]) maintain consistency without relying on locking certain parts of the shared document.

2.3 Access Control and Constraints

According to Sandhu et al., "*Access control* constrains what a user can do, thereby seeking to prevent activity that could lead to breach of security" [79]. Thus, it deals with the "elicitation, specification, maintenance, and enforcement of access control policies in software systems" [56, 79]. "An *Access control policy* is a high-level guideline that determines how accesses are controlled and access decisions determined" [79, 82]. An access or *authorization decision* is the decision if access to secured resource shall be granted or not. Put simply, access control concerns defining and enforcing *who* shall be granted to do *what* within a software system.

Access control policies can be divided into *authorization policies* and *obligation policies* [82, 83]. Authorization policies concern the rights of users, i.e., what users are permitted or not permitted to do. The essence of an authorization policy can be formalized using the triplet $\langle subject, object, operation \rangle$, whereby *subject* uniquely identifies a particular user (e.g., *Patrick Gaubatz*), *object* uniquely identifies a particular entity or artifact within a software system (e.g., *Thesis*) and *operation* describes an operation that can be performed on the corresponding *object* (e.g., *write*). This exemplary authorization policy grants *Patrick Gaubatz* the permission to *write* (his) *Thesis*. Authorization policies can be positive, i.e., granting permission to do something, and negative, i.e., denying permission to do something. On the other hand, obligation policies concern the duties of users, i.e., what users must or must not do when certain events occur. Obligation policies are event-triggered and define the activities users must perform on objects [15, 82, 83]. For instance, there exists an obligation policy that requires *Patrick Gaubatz* to *write* (his) *Thesis*, given the preceding event *PhD started*.

In recent years, *role-based access control* (RBAC) [29, 77] has become a de-facto standard for defining and enforcing security/access control policies in both research and industry. It supersedes *mandatory access control* (MAC, see, e.g., [41, 55]) and *discre-*

tionary access control (DAC, see, e.g., [6, 80]) in terms of flexibility. Thus, RBAC can be used to implement both MAC [69] and DAC [78]. In the context of RBAC, *roles* are used to model different job positions and scopes of duty within an information system. These roles are equipped with the *permissions* to perform their respective work tasks. Human users and other active entities (i.e., *subjects*) are assigned to roles according to their work profile [86, 87]. Thereby, subjects acquire all permissions necessary to fulfill their duties via their role memberships. RBAC has been extended and adapted to fit into many different application domains. For instance, process-related RBAC models (see, e.g., [39, 88, 90, 98]) enable the definition of permissions for the tasks that are included in (collaborative) processes. Similar extensions have been proposed that allow for securing Web resources or Web services (see, e.g., [1, 28, 52, 100]).

Besides customizing and adapting RBAC into different application domains, it has also been the foundation for several conceptual extensions. In particular, different types of *(access) constraints* have been proposed as a means for defining, e.g., *how* or *when* particular subjects may exercise particular permissions. For instance, *context constraints* are used to integrate context information (such as as temporal or spatial context of a user, see, e.g., [4, 13, 89]). Such constraints can, for instance, be used to make authorization decisions dependent on the current location of a particular user (i.e., is the user currently in the office?). Another important family of constraints have been generalized under the term *(task-based) entailment constraints* (see, e.g., [5, 8, 11]). A (task-based) entailment constraint places some restriction on the subjects who can perform a task x given that a certain subject has performed another task y. They cover two important concepts: *separation of duty* and *binding of duty*. Separation of duty can be enforced by *static mutual exclusion* (SME) and *dynamic mutual exclusion* (DME) constraints. A SME constraint defines that two tasks must never be assigned to the same role and must never be performed by the same subject (i.e., to prevent fraud and abuse). This constraint is global with respect to all instances of a particular process type. In contrast, DME refers to individual instances and can be enforced by defining that two tasks must never be performed by the same subject in the same process instance. SME or DME constraints provide means for enforcing a four-eyes-principle, which is intended to prevent fraud, abuse and error. In contrast to mutual exclusion constraints, binding constraints define that two bound tasks must be performed by the same entity. In particular, a *subject-binding* constraint defines that the same individual who performed the first task must also perform the bound task(s). Similarly, a *role-binding* constraint defines that bound tasks must be performed by members of the same role but not necessarily by the same individual. Such restrictions are required, e.g., in privacy critical environments, such as the e-health domain, where users acquire confidential knowledge.

Chapter 3

Problem Statement

The following section outlines the research problem and clarifies its domain and context. Secondly, we formulate the core research questions Section 3.2.

3.1 Problem Domain and Context

A collaborative process is a structured or unstructured process where two or more different stakeholders are working together to fulfill a shared, collective and bounded goal. In general, crucial "ingredients" of successful collaboration are coordination, clarity of roles, rights and responsibilities of each participating stakeholder (see, e.g., [59, 76, 91]). On the contrary, unclear role specification, i.e., not specifying *who* is supposed to do *what* within a collaborative process, may create dysfunctional ambiguity and conflict in an organization (see, e.g., [2, 7, 103]). Hence, as Zhu et al. noted, *an ideal collaborative system allows users to know and fulfill their obligations while respecting the rights and authority of other users in collaboration* [103]. Similarly, Ellis et al. argued that effective access control is an important means for preventing user-to-user interference [22].

This inherent need for defining the rights and responsibilities of stakeholders within collaborative processes is intensified by the fact that collaborative processes are often subject to security policies, business rules, laws and regulations. Numerous regulations and IT standards exist that pose compliance requirements for the corresponding systems. In particular, IT systems must comply with laws and regulations such as the Basel II/III Accords, the International Financial Reporting Standards (IFRS), the Health Insurance Portability and Accountability Act (HIPPA), or the Sarbanes-Oxley Act (SOX). For instance, one important part of SOX compliance is to provide adequate support for definition and enforcement of process-related security policies (see, e.g., [10, 14, 62]).

Role-based access control (RBAC) [29, 77] has become the de-facto standard for defining and enforcing such process-related security/access control policies in both research and industry. In particular, a process-related RBAC model (see, e.g., [88, 98]) enables

the definition of permissions for the tasks that are included in (collaborative) processes. Simply put, a process-related RBAC model is used to define *who* (i.e., subjects and roles) can do *what* (i.e., tasks and actions) within a particular process. In addition, different types of access constraints have been proposed as a means for defining, e.g., *how* or *when* particular subjects may exercise particular permissions. For instance, context constraints are used to integrate context information (such as as temporal or spatial context of a user, see, e.g., [4, 13, 89]). Also, entailment constraints are an important means to assist the specification and enforcement of compliant business processes (see, e.g., [5, 8, 11]). In particular, separation of duty constraints (i.e., DME and SME) provide means for enforcing a four-eyes-principle, which is intended to prevent fraud and error – something that is particularly important, e.g., in the context of SOX [62]. On the other hand, binding of duty constraints (i.e., subject-binding and role-binding) are often required in privacy critical environments, such as the e-health domain, where users acquire confidential knowledge. In summary, it can be noted that context constraints and entailment constraints, as well as domain-specific security constraints (see, e.g., [44]) are used to satisfy complex requirements that stem from laws, regulations or any other compliance rules.

Defining such access constraints and access control policies can be considered to be a translation process. More specifically, it involves the conversion of the various rules, regulations, and requirements into a formal set of access constraints and access control policies. This is usually a non-automatic, manual work task. Although domain experts, such as security experts or bank clerks, know the rules and regulations of their respective domain best, the formalization is usually done by software developers. Communication between software developers and the domain experts is literally the Achilles' heel in this particular situation. Unfortunately, there is usually a fundamental communication barrier between domain experts and software developers [27]. Software developers often have only a very vague idea about the domain-specific rules and regulations, while domain experts are most likely not familiar with any sorts of software development techniques or modeling methodologies required for formalizing access constrains. Due to misunderstandings or misinterpretations a considerable amount of additional time and effort has to be invested into clarifying discussions. In the worst case, it may even compromise a software system's security mechanism, i.e., allowing an unauthorized user to access secured resources [12, 50, 101]. Obviously, preventing such security breaches is of topmost importance. In summary, it can be noted that the inherent communication mismatch between security/domain experts and software developers may considerably add to the overall development and maintenance effort of defining access control policies.

As Xiao et al. noted, the second major issue in access control and security, besides incorrectly formalizing policies, is incorrect enforcement of the latter in the software

system's implementation [101]. One particular reason for such incorrect implementations is that access control enforcement as well as access control in general can be considered to be a prime example for a so-called cross-cutting concern. According to Kiczales et al. a concern is said to be cross-cutting when its implementation is scattered across the program and possibly tangled with the source code related to other concerns [53]. Several empirical studies (see, e.g., [9, 33, 42, 57, 102]) provide evidence that cross-cutting concerns often lead to a degradation of a software system's code quality. Eaddy et al. even provide empirical evidence suggesting that cross-cutting concerns effectively *cause* software defects [19]. Hence, we conclude that access control enforcement itself is a complex and error-prone concern from a software developer's point of view.

The situation gets worse if we consider the diverse variety of collaborative process types, which ranges from structured and rigid business processes to unstructured and flexible real-time collaborative Web applications. While business processes and process-aware information systems are omnipresent in the corporate contexts, real-time collaborative Web applications such as Google Docs, Etherpad, or Creately are getting more and more popular. This means that an ever increasing number of collaborative processes get realized as real-time collaborative Web applications. From a software developer's point of view, each type of process requires custom, domain-specific enforcement mechanisms and comes with its own set of technical challenges. In particular, large-scale Web applications require scalable and computationally efficient enforcement mechanisms. Another ongoing major trend that must be considered in this context is the shift towards mobile computing and mobile devices. As real-time collaborative Web applications proliferate, more and more users are going use mobile devices to participate in collaborative processes. Summing up, it can be noted, that the implementation of access control enforcement mechanisms is complicated by the diversity of collaborative process types and execution environments. Due to the ever increasing importance of Web applications and mobile devices, in this thesis, special attention is paid to these execution environments.

3.2 Research Questions

Based on the problem context defined in the previous section, this thesis aims to address three research questions, which are introduced and discussed in the following:

Research Question 1

How can the development and maintenance effort of defining different types of access constraints and implementing the respective enforcement mechanisms in different types of collaborative processes be reduced?

Research Question 1 concerns two important dimensions that contribute to the overall development effort of introducing and maintaining access control mechanisms in a software system, namely the definition of access constraints and the enforcement of the latter at runtime. Defining and maintaining access constraints typically does not only involve software developers, but also non-technical stakeholders, such as domain experts or security experts. Involving and coordinating both software developers and domain experts naturally increases the overall effort that is required in the formalization process. As has been discussed previously, the runtime enforcement of access control is a cross-cutting concern. As such, it is inherently prone to increase the complexity of software systems, degrade the code quality, and even cause software defects [19]. Repairing these defects, working against the degradation of code quality and coping with the ever-increasing complexity inevitably increases the development effort that is required to maintain such systems. Thus, a central goal of this thesis has been the attempt to reduce the effort that is needed to develop means for defining and enforcing access constraints in different types of collaborative processes.

Research Question 2

How can different types of access constraints in different types of collaborative processes be enforced in an effective, efficient and scalable way?

Research Question 2 is motivated by the observation that access control is really a key concern in many different application domains. If access control is a key concern the respective domain usually requires that the constrained system *effectively* enforces the defined policies and access constraints. This means that unauthorized access to secured resources must be prevented and it must therefore not be possible to somehow circumvent the respective enforcement mechanisms. While ensuring the effectiveness of enforcement mechanisms is certainly the topmost goal, some types of collaborative processes, such as real-time collaborative Web applications, require the *efficiency* and *scalability* of the enforcement mechanisms to be taken into consideration too. This is due to the fact

that users of such applications usually expect instantaneous update behavior. Therefore, enforcement mechanisms in such environments have to be efficient enough to not impose a perceivable performance penalty on the application's update behavior. Because in the Web context it is not uncommon having to deal with thousands of clients, being connected to the same Web application simultaneously, scalability is also a crucial requirement.

Research Question 3

How can we guarantee the consistency of access constraint models and the corresponding collaborative processes, especially considering constraint model changes at runtime and scenarios of offline use?

Research Question 3 is primarily driven by the ever increasing importance of Web applications, mobile computing and mobile devices. In particular, mobile computing demands solutions that enable users to continue working on access constrained, collaborative processes if a reliable network connection can not be guaranteed. Such unreliable network connections frequently occur if the user is on an airplane, in a train, in a basement, or in a rural area. In spite of this, most (real-time collaborative) Web applications do not adequately handle or do not consider such offline use scenarios at all. Besides that, it can be noted that especially large-scale Web applications are expected to be always available. Maintenance downtimes, due to updates or configuration changes, such as changing access constraint models, shall be avoided or at least kept as short as possible. Changing access constraint models dynamically at runtime would be the best solution. Unfortunately, due to the interrelations and dependencies between collaborative processes and their corresponding access constraint models as well as the immanent complexity of some access constraint models itself, such runtime changes inevitably lead to inconsistencies.

Research Question 4

What are the benefits and limitations of automatic enforcement of access constraints for collaborative processes?
How can we measure or quantify the impact of automatic enforcement of access constraints for collaborative processes?

Research Question 4 aims at finding evidence for the (beneficial) impacts of automatic enforcement of access constraints for collaborative processes. Besides discussions of lessons learned, we quantitatively study the efficiency and effectiveness of users. Such quantitative evidence can help decision-makers to conduct a cost-benefit analysis by estimating both the costs and the potential benefits of implementing automatic enforcement.

Chapter 4

Research Results and Contributions

The following section discusses the pursued research method. Section 4.2 lists all scientific papers that have been published and Section 4.3 provides a detailed overview of the scientific contributions that have been made in the course of this thesis.

4.1 Design Science Research Method

This thesis is founded on the principles of design science research. Design science research produces rigorous, meaningful results for information systems and gives the potential to investigate new technologies and to advance accepted practice – in the absence of a strong theory base – through the construction and evaluation of these systems and their components [92]. According to Peffers et al., the design science process includes six steps [71]: (1) problem identification and motivation, (2) definition of the objectives for a solution, (3) design and development, (4) demonstration, (5) evaluation, and (6) communication. Figure 4.1 illustrates and describes the artifacts that have been produced in the first two steps according to Peffers et al.

4.2 Publication Overview

This thesis consists of work that has either been published in scientific conferences, workshops, journals and books already, or is currently under review. The following list provides a detailed list of publications that are included in this thesis.

- **Paper A**: W. Hummer, P. Gaubatz, M. Strembeck, U. Zdun, and S. Dustdar. An Integrated Approach for Identity and Access Management in a SOA Context. In *16th ACM Symposium on Access Control Models and Technologies*, Innsbruck, Austria, June 2011

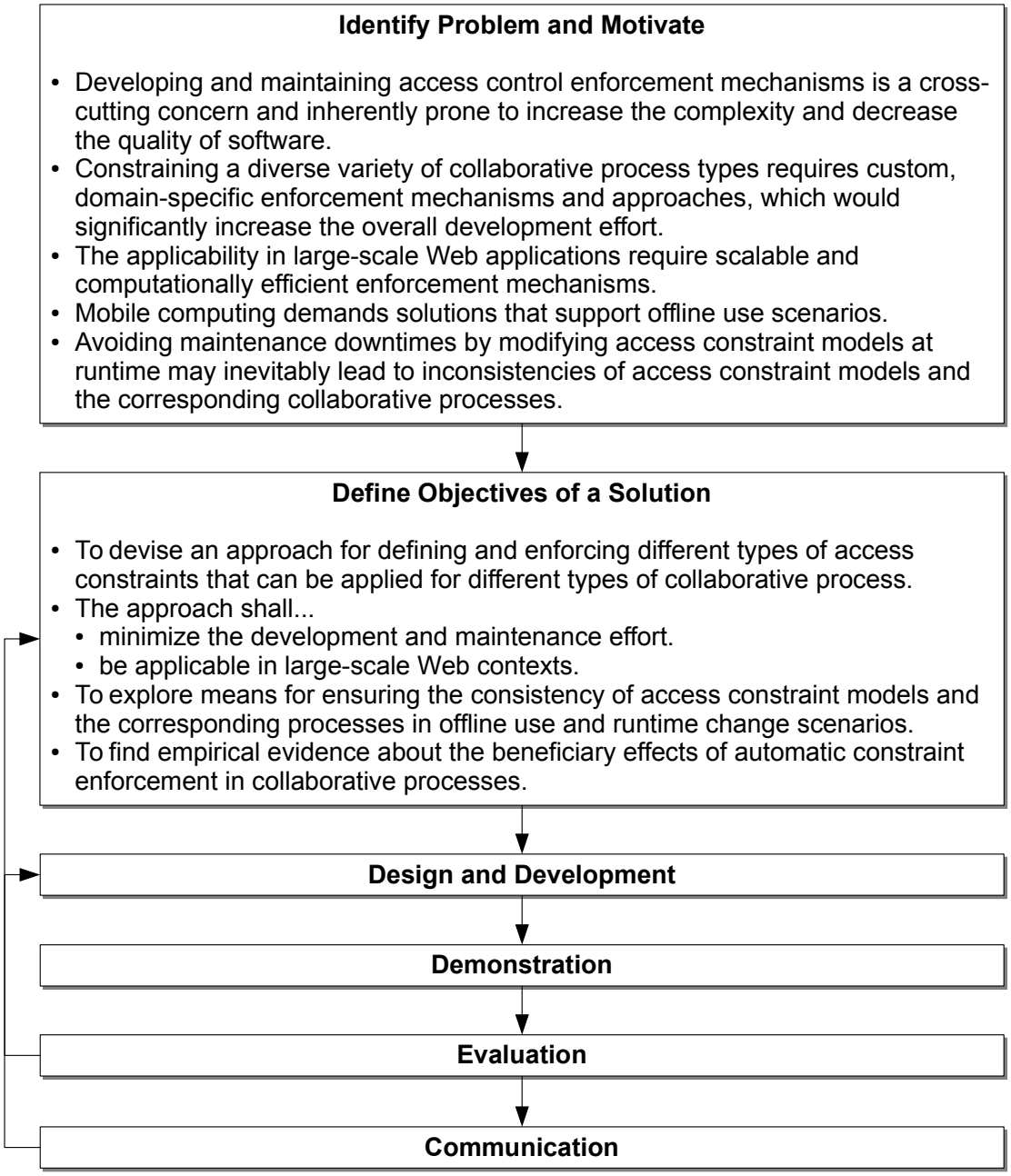


Figure 4.1: Applying the first two steps of Design Science according to Peffers et al. [71]

- **Paper B**: W. Hummer, P. Gaubatz, M. Strembeck, U. Zdun, and S. Dustdar. Enforcement of Entailment Constraints in Distributed Service-based Business Processes. *Information and Software Technology*, 55(11), November 2013

- **Paper C** T. Quirchmayr, P. Gaubatz, M. Strembeck, and U. Zdun. Consistency Checking and Resolution Strategies for Runtime Conflicts Resulting from Changes in Process-related RBAC Models. submitted to *Advances in Verifiably Secure Process-aware Information Systems*, submitted in June 2014

- **Paper D**: P. Gaubatz and U. Zdun. UML2 Profile and Model-Driven Approach for Supporting System Integration and Adaptation of Web Data Mashups. In *4th International Workshop on Lightweight Integration on the Web*, Berlin, Germany, July 2012

- **Paper E**: P. Gaubatz and U. Zdun. Supporting Entailment Constraints in the Context of Collaborative Web Applications. In *28th Symposium On Applied Computing*, Coimbra, Portugal, March 2013

- **Paper F**: P. Gaubatz, W. Hummer, U. Zdun, and M. Strembeck. Supporting Customized Views for Enforcing Access Control Constraints in Real-time Collaborative Web Applications. In *13th International Conference on Web Engineering*, Aalborg, Denmark, July 2013

- **Paper G**: P. Gaubatz, W. Hummer, U. Zdun, and M. Strembeck. Enforcing Entailment Constraints in Offline Editing Scenarios for Real-time Collaborative Web Documents. In *29th Symposium On Applied Computing*, Gyeongju, Korea, March 2014

- **Paper H**: I. Lytra, P. Gaubatz, and U. Zdun. Two Controlled Experiments on Model-based Architectural Decision Making. submitted to *Information and Software Technology*, submitted first revision in January 2015

- **Paper I**: P. Gaubatz, I. Lytra, and U. Zdun. Automatic Enforcement of Constraints in Real-time Collaborative Architectural Decision Making. *Journal of Systems and Software*, accepted for publication in January 2015

Figure 4.2 puts these publications into their respective contexts. For instance, Paper A presents "an integrated approach for identity and access management in a SOA context" and is therefore placed within the contexts Business Processes, Access Control and Constraints and Model-driven Development. Similarly, Paper D introduces a model-driven approach for Web data mashups and therefore overlaps both Model-driven Development and Web Applications.

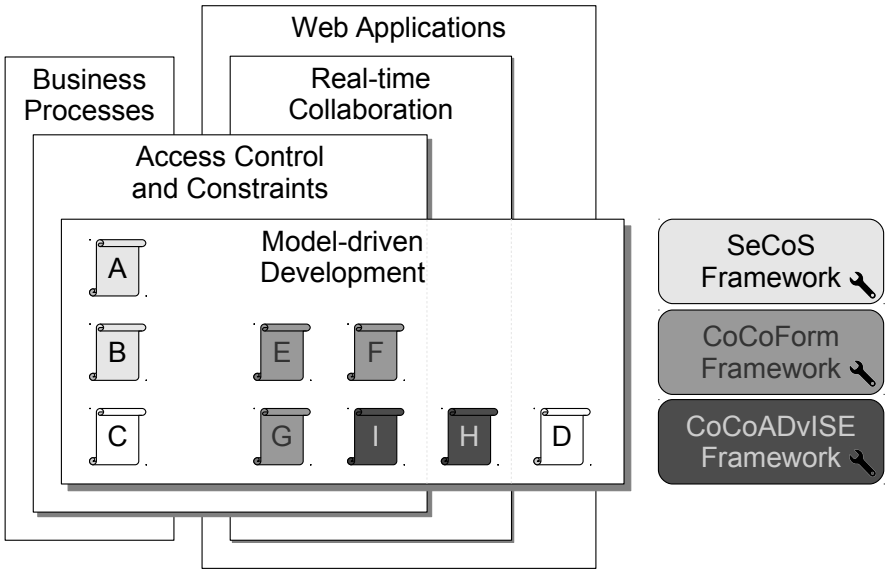


Figure 4.2: Context of Publications and Developed Prototypes

Besides indicating the context of each publication, Figure 4.2 also names the three prototypical implementations (i.e., the SeCoS, CoCoForm and CoCoADvISE Frameworks) that have been developed in the course of this thesis. As each prototype takes a different shade of gray, we can easily tell which prototype has been presented in which particular publication. For example, the CoCoADvISE Framework (i.e., dark gray) is part of Paper H and Paper I.

4.3 Scientific Contributions

In this section we want to highlight and discuss the main scientific contributions that have been published within the scope of this thesis. Figure 4.3 provides an overview of the five main contributions, clearly indicating that Contribution 1 sets the boundaries and provides the foundation for Contribution 2, Contribution 3 and Contribution 4 while Contribution 5 is an orthogonal aspect of the first four contributions.

Contribution 1

Model-driven approach for defining and enforcing different types of access constraints in different types of collaborative processes.

We have designed and developed a model-driven approach for defining and enforcing access constraints in collaborative processes. More precisely, our approach has been applied for two different types of access constraints, i.e., constraints from the RBAC context and collaborative decision making constraints. The approach has also been extended and applied for the following three different types of collaborative processes:

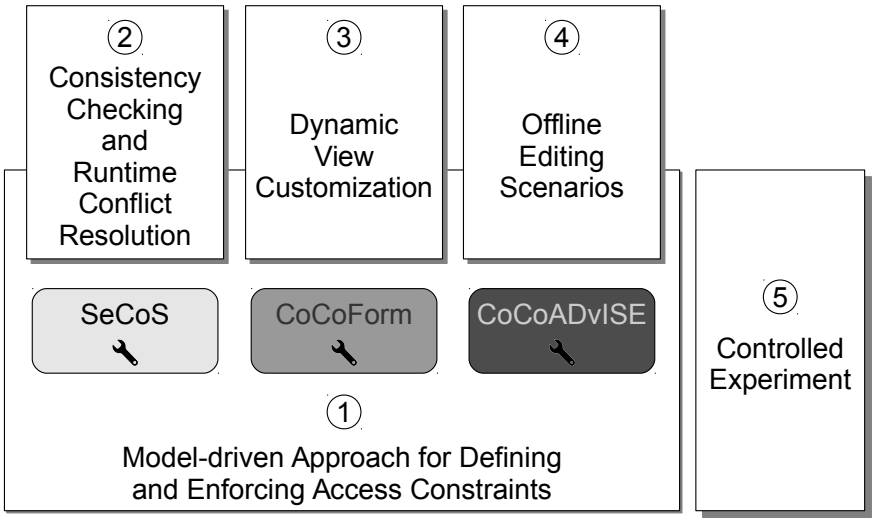


Figure 4.3: Overview of Scientific Contributions

a. **Business processes in distributed service-based architectures.** In this context, we introduced the SeCoS framework. The purpose of this framework is to secure service invocations within a business process by making them subject to access control policies and constraints. Our approach incorporates the concept of RBAC, entailment constraints and single sign-on. A custom domain-specific language (DSL), which is an important cornerstone of our approach, provides means for defining RBAC policies and entailment constraints. In addition, it allows for tagging service invocations within process definitions with special security annotations. At deployment time, the control flows of these annotated process definitions are automatically augmented with additional code for enforcing the defined RBAC policies and constraints. At runtime, this enforcement logic requests authorization decisions from the central Policy Decision Point (PDP) service, used for determining whether the invoking user shall be granted the permission to invoke the respective service, or not. The inclusion of the single sign-on concept, guarantees applicability of our approach in larger cross-organizational environments, enabling one organization's users to access secured services of another trusting organization. Finally, we performed extensive evaluations of our SeCoS prototype implementation. In particular, we verified the consistency and quantified the performance penalty of our proposed enforcement mechanism.

b. **Real-time collaborative form editing.** In this context, we introduced the CoCoForm framework. This frameworks aims to be a straightforward model-driven approach for specifying and enforcing access constraints (such as entailment constraints) in real-time collaborative Web applications. We demonstrated the applicability of this approach using a real-time collaborative Web form editing application. We contrast such real-time collaborative Web applications with "traditional" form-based business

applications, where users are confronted with standardized forms and precisely specified form fields. In such business applications, the structure of the forms is often hard-coded into custom-made legacy applications. An alternative solution would be workflow- or pageflow-based applications, where the form consists of workflow tasks to be executed in a precisely prescribed order. Beside cluttering the control flow with additional code concerning access control enforcement, these business applications exhibit a major disadvantage: Their control flows are statically prescribed at design time and it is not possible to leave this prescribed path. On the contrary, CoCoForm does not prescribe any execution path at all. In CoCoForm, the structure of forms is modeled using a custom meta-model and each element within a form can be made subject to various access constraints. This allows form fields to be filled out concurrently by various users at the same time and can easily accommodate "unforeseen" deviations from the originally intended workflows. CoCoForm is a generic approach that can be applied to many real-time collaborative Web applications. It requires some modifications to existing Web applications, but these – as well as the generation of all other required artifacts – can optionally be automated with model-driven development techniques. Analogous to the SeCoS framework, in CoCoForm the enforcement logic delegates authorization decisions to a central, service-based PDP too.

c. **Real-time collaborative decision making and documenting.** In this context, we introduced the CoCoADvISE framework. Our approach is the first one to consider the precise definition and automatic enforcement of constraints in real-time collaborative architectural decision making. It includes a formal meta-model containing a set of novel decision making constraints including precisely defined semantics of each constraint type. Conceptually, the CoCoADvISE framework is quite similar to CoCoForm. That is, instead of making form elements subject to various access constraints (as in CoCoForm's case), in CoCoADvISE, reusable decision models are made subject to decision making constraints. At runtime, these constrained decision models are automatically transformed into Web-based questionnaires (similar to CoCoForm's Web forms). Questionnaires enable multiple, possibly geographically dispersed software architects and stakeholders to participate in the group decision making and documentation process, while the constraint enforcement mechanisms guarantee compliance to the defined decision making constraints.

Figure 4.4 illustrates the main components of our model-driven approach in an abstract, high-level view, as it is used in all three types of collaborative processes. Our approach is based on three major concepts: model-driven development, separation of concerns, and service-orientation. In the following we explain how each of these concepts is used in our approach to define and enforce access constraints in collaborative processes:

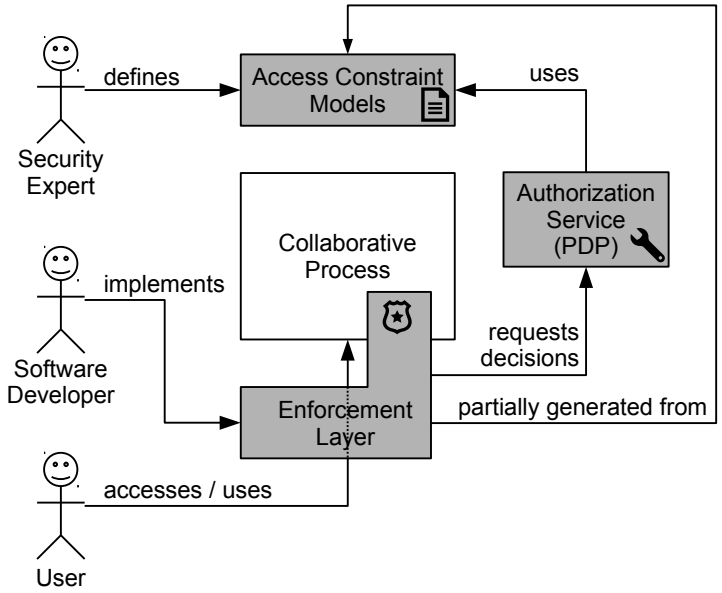


Figure 4.4: Defining and Enforcing Access Constraints in Collaborative Processes

1. **Model-driven Development.** The model-driven development concept (see, e.g., [3, 60, 81]) proposes the *model as the primary artifact* in software development. We adhere to this concept by formalizing various types of – potentially domain-specific – access constraints in the form of Access Constraint Models. We also propose the development of DSLs, as a means for defining Access Constraint Models. This has the benefit that a potentially larger group of people may be involved in the formalization process. Eventually, we aimed for empowering non-software-developing people, such as Security Experts, to define and maintain access control related policies.

   Another essential concept of model-driven development is *model verification*. Especially in our context, i.e., security and access control, model verification can be a valuable tool. This is due to the fact, that access control policies may contain various inconsistencies, conflicting access constraints (such as entailment constraints), or even loopholes that may – in the worst case – allow attackers to circumvent access control mechanisms. Model verification techniques, although being beyond the scope of this thesis' contributions, can mitigate such problems to a large extent by propagating model invariants to be defined. Checking these model invariants eventually exposes inconsistencies, conflicts or loopholes within the models.

   The final step in a model-driven development environment is usually a combination of *model transformation* and *code generation*. Correspondingly, in our approach the Access Constraint Models are semi-automatically transformed (i.e., code has to be annotated manually) into executable code that is part of the Enforcement Layer. Code generation is important, because enforcement code can be considered to be "boilerplate

code" and manually writing code would be both time-consuming and error-prone. In addition to code generation, certain components in our proposed system architecture, such as the central Authorization Service or Policy Decision Point, may also interpret and manipulate parts of the Access Constraint Models dynamically at runtime.

2. **Separation of Concerns.** The concept of Separation of Concerns (see, e.g., [17, 49, 70]) advises that software should be decomposed in such a way that different concerns or aspects of the problem at hand are solved in well-separated modules or parts of the software [16]. Security and access control is a prime example of a particular "concern" as in "separation of concerns". Consequently, the general idea of our approach is to decouple the Enforcement Layer from the actual application code of the Collaborative Process, as much as possible. Access Constraint Models are decoupled from the application code in a similar fashion. And finally, our approach also includes the idea of strictly separating the concern of authorization decision making from the actual enforcement of authorization decisions. Figure 4.4 clearly illustrates the application of the Separation of Concerns principle. While Access Constraint Models, the Enforcement Layer and the Authorization Service are cleanly separated from each other, we can see, that the Enforcement Layer overlaps the Collaborative Process slightly. This means, that access control enforcement always requires adaptations and/or modifications to be made on the underlying application code. However, a key concern of our approach is to keep this overlap as small as possible.

3. **Service-orientation.** The Service-oriented Architecture (see, e.g., [23, 25, 54]) as well as the corresponding service-orientation paradigm aims for positioning services as the primary means through which application logic is represented [26]. A service is a loosely coupled and self-contained software unit that provides a particular functionality. In our approach, we adopt these principles by proposing a central, service-based Authorization Service or PDP (Policy Decision Point). In particular, this Authorization Service represents the single authority within a particular system that is empowered to make authorization decisions. This architecture allows for supporting the enforcement more sophisticated access constraints, such as entailment constraints, which requires such a Single Source of Truth to make authorization decisions. A single service can potentially also be reused for multiple instances of the same collaborative process. Finally, we found that a service-based Authorization Service naturally fits quite well into the context of today's service-based business processes or the context of real-time collaborative Web applications.

Contribution 1 has originally been presented in Paper A, Paper B, Paper D, Paper E and Paper I.

Contribution 2

Generic and reusable consistency checking and resolution strategies and algorithms for runtime conflicts resulting from changes in process-related RBAC models.

Contribution 2 is an extension of Contribution 1(a) and has originally been presented in Paper C. This contribution has gone some way towards enhancing our understanding of the effects and pitfalls of changing process-related RBAC models dynamically at runtime. This was driven by the desire to further establish the approach of viewing process-related RBAC models as long-lived but dynamic artifacts that are constantly subject to change, instead of completely static, "deploy once and never touch it again" type of artifacts. In particular, we have examined the impact of changes in process-related RBAC models on the respective process instances at runtime. We have systematically analyzed how every possible change operation might negatively affect the runtime consistency of process instances. For each potentially harmful change operation we have derived a generic and reusable conflict detection algorithm that detects runtime consistency conflicts and is independent of a certain software platform or programming language. Finally, we have tackled the issue of resolving runtime consistency conflicts by proposing generic resolution strategies that take the current state of conflicting task instances into account.

Contribution 3

Dynamic view customization, a novel concept for enforcing access control in large-scale, real-time collaborative Web applications, as well as the prototypical implementation and evaluation of this concept.

Contribution 3 is an extension of Contribution 1(b) and has originally been presented in Paper F. This contribution demonstrates that access control policies and constraints – in particular entailment constraints – in the context of real-time collaborative Web applications can effectively be enforced by dynamically constraining user interface (UI) elements for certain subjects. We call this process "view customization". Further, we show that our service-based approach can be used to realize the corresponding UI view configuration functionality and we provide evidence that it is potentially capable of meeting the – especially in the context of real-time collaborative Web applications important – requirement of nearly instantaneous update behavior, even for a large number of simultaneously connected users. Although the client-side part of the UI view configuration functionality is built upon the Model-View-ViewModel pattern [84], we show that it can easily coexist with others. In fact, we argue that our approach is complementary to currently available frameworks and solutions that support the development of real-time collaborative Web applications, because it is completely decoupled from the collaborative

aspects of the application. In essence, supporting view customization merely requires the deploying a single, dedicated and self-contained View Service as well as hooking-in the View Updater code into the client-side application code.

Contribution 4

A novel approach for supporting offline-editing scenarios in the context of access constrained, real-time collaborative Web documents, as well as the prototypical implementation and evaluation of this approach.

Contribution 4 is an extension of Contribution 1(b) and Contribution 3 and has originally been presented in Paper G. Existing solutions for access control enforcement typically rely on a central service, the Policy Decision Point (PDP). However, for use cases with unreliable or limited connectivity, such as mobile devices, a permanent connection to this centralized PDP can not be guaranteed. This shortcoming lead us to devise a novel approach that enables users to locally edit access constrained, real-time collaborative Web documents while their devices are temporarily offline. Our CoCoForm prototype implementation demonstrates, that offline editing for access constrained, real-time collaborative Web documents can effectively be realized using a combination of client-side access control enforcement and a document merging approach. We highlighted that merging such documents is inherently prone to conflicts and motivated the need for a merge approach that is capable of detecting and resolving conflicts automatically. We provided evidence that many possible conflicts can be resolved automatically and that both the merge algorithms and our prototypical document merge approach work with acceptable runtime performance and scalability even for lots of simultaneous merge requests and documents with lots of data fields. We also argued that the prioritization of online performed changes of the Web document in favor of offline performed changes is crucial in the context of real-time collaboration. We introduced the concept of an offline weight as a means of discriminating offline performed changes in the merge process and discovered a tradeoff relationship inherent to this approach, i.e., a higher offline weight increases the automatability of the merge process but also increases the number of situations where online data fields have to be reverted. The optimal offline weight has to be determined empirically for a given document and the corresponding entailment constraint model.

Contribution 5

A controlled experiment that evaluates and quantifies the beneficiary effects of automatic constraint enforcement on the productivity of users.

Contribution 5 is an extension of Contribution 1(c) and has originally been presented in Paper I and Paper H. While the aforementioned contributions focus on aspects concerning the development of constrainable collaborative processes, Contribution 5 shifts to a completely different point of view, namely the perspective of users. With the help of controlled experiments using our CoCoADvISE tool, we were able to report strong evidence that the automatic enforcement of constraints leads to increased time and effort related efficiency and effectiveness of the users while making and documenting architectural decisions. In our experiment setup, we observed that the treatment group that was supported by CoCoADvISE's automatic constraint enforcement mechanisms, could finish nearly 16% more work tasks, requiring 41% less time and 44% less work steps, than the other treatment group, that was not supported by the automatic enforcement mechanisms. Finally, we consider our approach and accompanying CoCoADvISE tool to be relevant and useful for other collaborative software engineering tools as well, which involve various stakeholder roles and distributed teams.

Figure 4.5 summarizes the main research contributions described above. It complements Figure 4.1 by illustrating and describing the artifacts that have been produced in the last four steps of the design science research method according to Peffers et al.

In addition to these main contributions, the following contribution has been made in course of writing this thesis. In [85] we presented an exploratory experiment concerning the complexity of different types of API designs. In particular, we could report that in our experiment, a textual DSL exhibited a smaller API complexity than three other object-oriented frameworks – an aspect that is often claimed in literature, but rarely backed by empirical evidence. As the topic of this particular publication is somewhat out of scope of this thesis, we have excluded it from the latter.

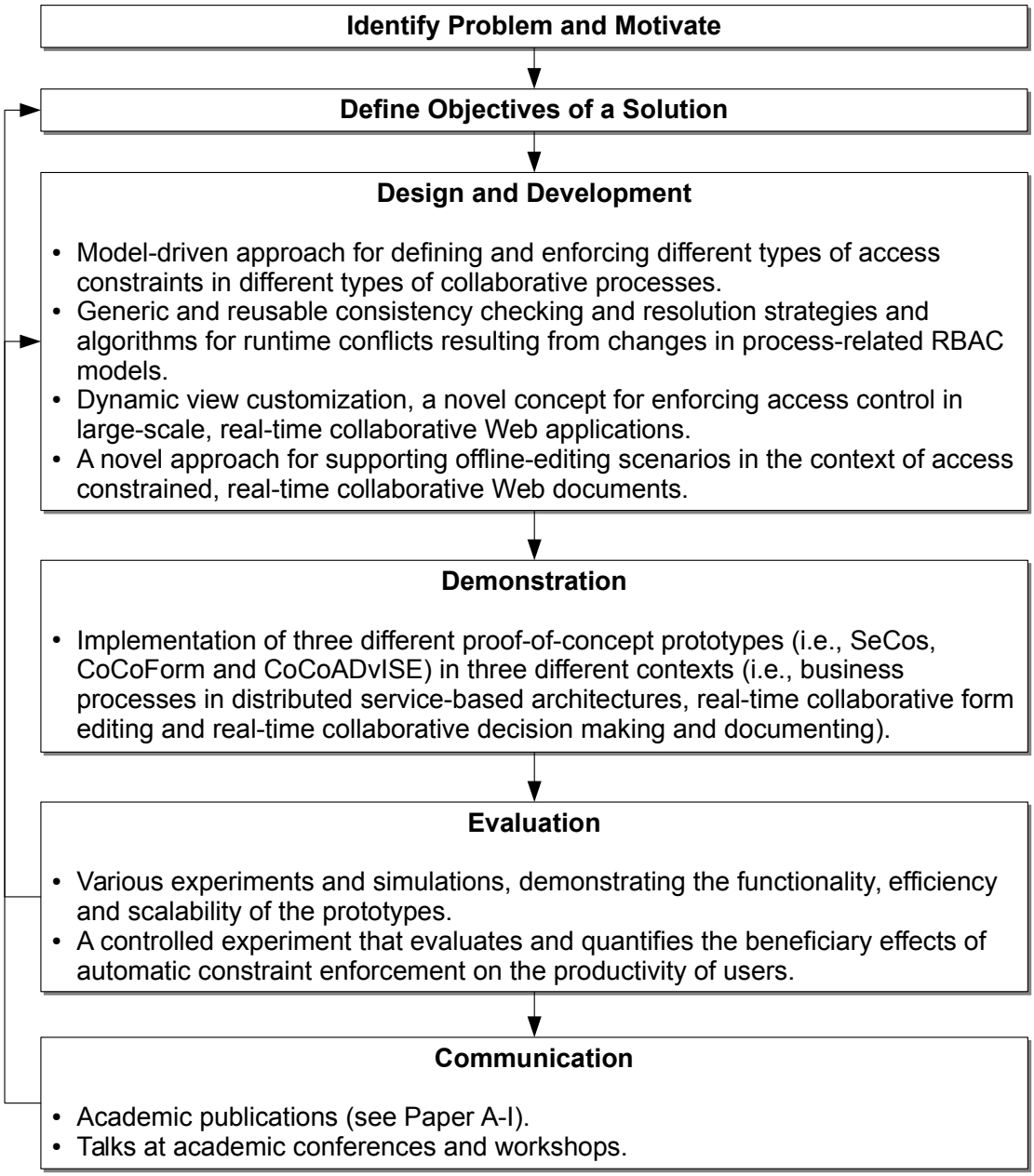


Figure 4.5: Applying the last four steps of Design Science according to Peffers et al. [71]

Chapter 5

Conclusions

In this final chapter, the research questions formulated in Section 3.2 are revisited and put into perspective with the list of scientific contributions elaborated in Section 4.3. Finally, Section 5.2 concludes with a discussion of open topics and potential future research.

5.1 Research Questions Revisited

Let us now revisit the three research questions, that have been formulated in Section 3.2. In this section, we will summarize how these central questions have been addressed by the scientific contributions discussed in the previous Section 4.3. Figure 5.1 provides an overview of the interrelations of Research Questions, Contributions and Publications.

Research Question 1: How can the development and maintenance effort of defining different types of access constraints and implementing the respective enforcement mechanisms in different types of collaborative processes be reduced?

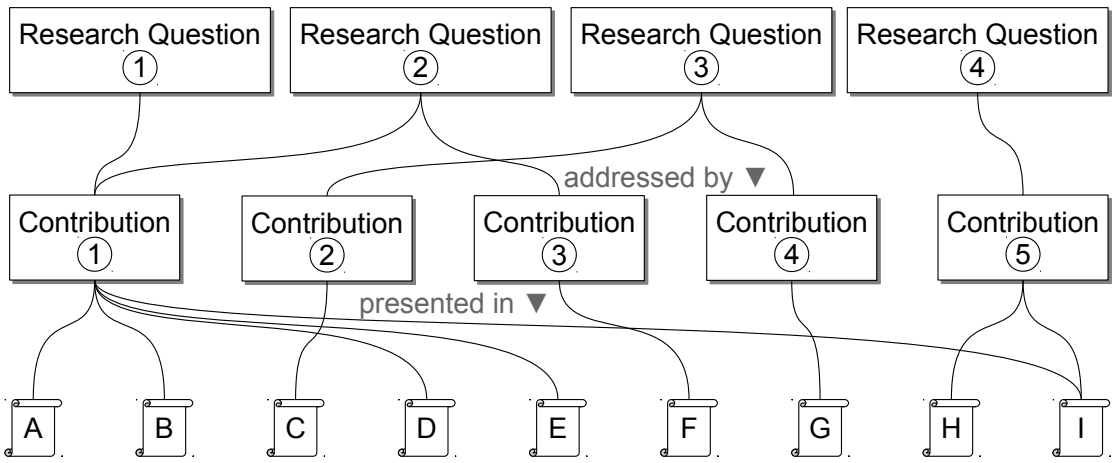


Figure 5.1: Overview of Research Questions, Contributions and Publications

This question has been addressed by Contribution 1. Our proposed model-driven development approach reduces both the development and maintenance effort by leveraging the following concepts and ideas. First, we propose the usage of models and DSLs as a means for formalizing access constraints and policies. Relying on concise meta-models accompanied by tailor-made DSLs empowers domain experts, such as security experts, to define and maintain access constraint models, thereby relieving the software developer's workload. Model verification techniques can reduce the workload even further, by exposing inconsistencies, conflicts or loopholes, which might potentially cause security breaches, within the access constraint models.

Automation, a cornerstone of model-driven development, is by far the most effective technological means for boosting productivity and reliability [81]. For this reason, in our approach, access constraint models are both, interpreted dynamically at runtime, and transformed into executable code. On the contrary, manually writing code would be both, time-consuming and error-prone, resulting in reduced productivity of software developers and reduced reliability of the security enforcements mechanisms. Especially in Paper A and Paper B we could demonstrate that code generation can greatly reduce the development effort in our context as compared to manual implementation. In Paper D, Paper E and Paper I we focus on model interpretation, instead of code generation.

Adhering to the concept of Separation of Concerns promises to increase the maintainability, i.e., it decreases the effort that needs to be invested in order to maintain the security enforcements mechanisms. This is especially important, because access control is a cross-cutting concern, that is inherently prone to increasing the complexity and degrading the quality of a software system's code. Consequently, our approach suggests decoupling security enforcement mechanisms and access constraint models from the actual application code, as much as possible. For instance, in Paper A and Paper B we propose an annotation mechanism as a non-intrusive way of linking access constraint models with process definitions. Besides annotation, the original process definition is left totally untouched. Hence, we argue, that this additional annotation step causes only a marginal and negligible increase in code complexity and degradation of code quality. Our central, service-based Authorization Service can potentially also increase the maintainability, as a single instance of the service can be reused for multiple instances of the same collaborative process.

Research Question 2: How can different types of access constraints in different types of collaborative processes be enforced in an effective, efficient and scalable way?

This question has been addressed by Contribution 1 and Contribution 3. Access control is really a key concern in many different application domains end enforcement mechanisms are essential in such situations. By implementing our three research proto-

types, SeCos (see Paper A and Paper B), CoCoForm (see Paper E, Paper F and Paper G) and CoCoADvISE (see Paper I and Paper H), that have been developed during the course of this thesis, we can assure the *effectiveness* of the proposed enforcement mechanisms. Thus, we could demonstrate that our enforcement mechanisms correctly enforce various types of access constraints at runtime.

Regarding the *efficiency* and *scalability* aspects of enforcement, we thoroughly evaluated our prototypes by conducting extensive performance experiments. In the context of business processes (see Paper A and Paper B), we illustrate that our enforcement mechanisms operate with an overhead that scale well up to the order of several ten thousand logged service invocations. Especially in the context of real-time collaborative Web applications (see Paper F and Paper G) we explicitly focus on evaluating the *efficiency* and *scalability* of our proposed dynamic view customization approach (see Contribution 3). In Paper F, we show that our dynamic view customization approach provides linear scalability and our prototype implementation can serve thousands of users, collaboratively working on the same Web document. Performance measurements reveal, that even in the case of 2000 simultaneously connected users, the average response time of our view (customization) service remains well below a second.

Research Question 3: How can we guarantee the consistency of access constraint models and the corresponding collaborative processes, especially considering constraint model changes at runtime and scenarios of offline use?

This question has been addressed by Contribution 2 and Contribution 4. While our model-driven approach comprising Contribution 1 concern both the development and maintenance aspects, in Contribution 2 we deliberately focus on maintenance. Based on previously documented observations (see e.g., [32, 52]) that RBAC can decrease both the administration and maintenance costs, in Paper C we start with the assumption of process-related RBAC models being treated as long-lived but dynamic artifacts that are constantly subject to change [30]. For the sake of consistency, changing process-related RBAC models is typically preceded by shutting down the corresponding process instances or even the whole process-ware information system. However, especially in the case of long-running processes such procedures are impractical. In Paper C we therefore propose a novel set of consistency checking algorithms that aim for allowing manipulating process-related RBAC models while the corresponding process instances are currently being executed. We show that performing changes at runtime may inevitably lead to runtime consistency conflicts within the corresponding access constraint models. We tackle this issue by complementing the proposed algorithms with generic conflict resolution strategies, that may be used to resolve conflicts (semi-) automatically. In summary, our consistency checking and resolution strategies can considerably reduce the

required maintenance effort in situations that demand changes of process-related RBAC models to be effected instantaneously. We consider the effort reduction to be substantial, because the alternative manual approach involves manually resolving runtime conflicts, which is error-prone, cumbersome and impractical for complex models.

While Contribution 2 concerns constraint models changes at runtime, in Contribution 4 our focus shifts towards scenarios of offline use. In Paper G we propose a novel approach that enables users to edit access constrained, real-time collaborative Web documents while their devices (e.g., notebooks or smartphones) are currently offline. This approach was driven by the fact that existing solutions for access control enforcement typically require permanent connection to central PDPs – something that can not be taken as granted in the context of mobile computing due to unreliable or limited network connectivity. Our approach combines client-side access control enforcement and document merging algorithms that are capable of detecting and resolving inevitable merge conflicts automatically. The merge algorithms are parameterizable with an offline weight. More precisely, we introduced offline weights as a means of discriminating offline performed changes of the collaborative Web document over online performed ones. In experimental simulations, we discovered that increasing the offline weight increases the automatability of the merge process, i.e., the chance that a merge can be performed automatically. For instance, in one exemplary scenario this chance more than doubles (from 20% to 47%) if we double the offline weight (from 0.4 to 0.8). However, increasing the offline weight also increases the number of merge conflicts, i.e., situations where online performed changes have to be reverted. In the same exemplary scenario and the same offline weight increase the chance of conflicts rises from 9% to 44%. Orthogonal to Research Question 2, which concerns the scalability of access constraint enforcement mechanisms, we conducted extensive performance experiments that aim for evaluating the scalability of our proposed merge mechanism. Our results indicate linear scalability and even in the case of 500 users, simultaneously submitting their offline performed changes, the average response time of our (offline) merge service remains below a tenth of a second. Also considering the results of our evaluation, concerning the memory consumption and execution time of merge algorithms and snapshotting approaches, we conclude, that our proposed approaches are sufficiently efficient to be applicable in real-world scenarios and large-scale Web applications. In summary, Contribution 4 demonstrates how offline use scenarios can be supported in a way that access constraint models and the corresponding collaborative process are kept in a consistent state.

Research Question 4: What are the benefits and limitations of automatic enforcement of access constraints for collaborative processes? How can we measure or quantify the impact of automatic enforcement of access constraints for collaborative processes?

This question has been addressed by Contribution 5. Driven by the fact that implementing access constraint enforcement mechanisms is always a substantial investment, we aimed for finding means for quantifying the prospective benefits of such an investment. In Paper H and Paper I we present the results of two controlled experiments. In the context of collaborative architectural decision making and documenting architectural decisions, we could provide strong evidence that automatic enforcement of constraints increases both the efficiency and effectiveness of users. In the experiment, we found that users completed their assigned work tasks faster (i.e., requiring 41% less time) while at the same time having to invest less effort (i.e., requiring 44% less work steps). This is because automatic enforcement mechanisms take away the burden of detecting, preventing and resolving constraint violations "manually" from the user. We found that our linear regression model presented in Paper I serves as a good estimator for predicting the effort reduction to be expected when introducing automatic enforcement mechanisms to a collaborative process. We argue that predicting and quantifying the potential effort reduction is a crucial instrument in outweighing the costs and benefits of implementing enforcement mechanisms. Although it is possible that the results of our experiments are specific to collaborative decision making and documenting architectural decisions, we believe that virtually any kind of collaborative process that concerns different stakeholder roles and demands to be restricted by various domain and context specific constraints will benefit from automatic constraint enforcement in a similar way to CoCoADvISE.

5.2 Future Work

The research presented in this thesis raised several questions and unlocked a number of important challenges which were beyond the scope of this thesis. In particular:

- Although we have started to explore the implications of changing access constraint models dynamically at runtime, our proposed model-driven approach still assumes and requires static process/document models, which is a major limitation in certain contexts. Especially in the case of ad-hoc processes, such as collaborative rich text editing (e.g., Google Docs), our approach would have to be adapted and/or extended to be able to deal with such completely dynamic process/document models.

- SeCoS, our prototype implementation for securing business processes still has limitations. For instance, in the case of highly parallel processing logic, advanced synchronization mechanisms would be required for ensuring compliance to the defined access constraints. Moreover, the query mechanism that checks access constraints for validity needs to be further optimized for very large log data sets (in the order of millions of invocations). We envision advanced data storage and compression tech-

niques, as well as optimized query mechanisms to further reduce this increase of overhead over time. Also, it would be interesting to investigate the use of additional security annotations, in order to analyze the generalizability of our approach.

- Our work on changing access constraint models dynamically at runtime has thrown up questions and open issues in need of further investigation. For instance, it would be interesting to study the conflict resolution's possible degree of automation. In particular, we would need a reliable instrument for automatically choosing the most sensible resolution strategy for a given runtime consistency conflict at hand. Future work might complement our work by deriving similar conflict detection algorithms for access constraints other than task-based mutual exclusion constraints and binding constraints.

- Our work on client-side access control enforcement and the CoCoForm prototype revealed several limitations inherited from HTML5 and Web browser implementations, such as the limited client-side storage capacity which be problematic if we have to deal with huge document and access constraint models (i.e., tens of thousands of model elements). Our initial experiments with two different document merge approaches leads us to believe that there is still room left for improvements in that area. More precisely, we think that more sophisticated merge approaches could further increase the automatability without having to sacrifice and revert already performed work. Another interesting topic would be to devise an approach for estimating (i.e., instead of determining it empirically) the optimal offline weight for a given document and the corresponding entailment constraint model.

- More empirical evidence about the supportive effect of automatic enforcement of constraints in collaborative architectural decision making tools on the efficiency and effectiveness of users, should be collected. Also, our assumptions regarding the supportive effects of automatic constraint enforcement, should be tested with practitioners, in order to receive feedback concerning the usability of our CoCoADvISE tool. Our approach should also be tested with different group sizes, in different system domains, and with different decision models. Abstracting from collaborative architectural decision making and conducting similar studies and experiments in other contexts, such as collaborative text-editing (i.e., CoCoForm), could provide provide evidence, that our results are generalizable and valid for virtually any kind of real-time collaborative process.

Bibliography

[1] G.-J. Ahn, R. Sandhu, M. Kang, and J. Park. Injecting rbac to secure a web-based workflow system. In *Proceedings of the Fifth ACM Workshop on Role-based Access Control*, RBAC '00, pages 1–10, New York, NY, USA, 2000. ACM.

[2] B. E. Ashforth. *Role transitions in organizational life: An identity-based perspective*. Lawrence Erlbaum Associates, 2001.

[3] C. Atkinson and T. Kuhne. Model-driven development: a metamodeling foundation. *Software, IEEE*, 20(5):36–41, 2003.

[4] E. Bertino, P. A. Bonatti, and E. Ferrari. Trbac: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, 4(3):191–233, Aug. 2001.

[5] E. Bertino, E. Ferraria, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, 2(1):65–104, 1999.

[6] K. J. Biba. Integrity considerations for secure computer systems. Technical Report ESD-TR-76-372, USAF Electronic Systems Division, Bedford, MA, Apr. 1977.

[7] R. P. Bostrom. Role conflict and ambiguity: Critical variables in the mis user-designer relationship. In *Proceedings of the Seventeenth Annual Computer Personnel Research Conference*, SIGCPR '80, pages 88–115, New York, NY, USA, 1980. ACM.

[8] R. Botha and J. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3):666–682, 2001.

[9] N. Cacho, E. Figueiredo, R. Maranhao, A. Garcia, C. M. F. Rubira, et al. Exceptions and aspects: the devil is in the details. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 152–162. ACM, 2006.

[10] J. Cannon and M. Byers. Compliance Deconstructed. *ACM Queue*, 4(7):30–37, Septemper 2006.

[11] D. Clark and D. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *IEEE Symp. on Security and Privacy*, April 1987.

[12] M. Croitoru, L. Xiao, D. Dupplaw, and P. Lewis. Expressive security policy rules using layered conceptual graphs. In M. Bramer, F. Coenen, and M. Petridis, editors, *Research and Development in Intelligent Systems XXIV*, pages 237–250. Springer London, 2008.

[13] F. Cuppens and N. Cuppens-Boulahia. Modeling contextual security policies. *Int. J. Inf. Secur.*, 7(4):285–305, July 2008.

[14] M. Damianides. How does SOX change IT? *Journal of Corporate Accounting & Finance*, 15(6):35–41, 2004.

[15] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, POLICY '01, pages 18–38, London, UK, UK, 2001. Springer-Verlag.

[16] B. De Win, F. Piessens, W. Joosen, and T. Verhanneman. On the importance of the separation-of-concerns principle in secure software engineering. In *Workshop on the Application of Engineering Principles to System Security Design*, pages 1–10, 2002.

[17] E. W. Dijkstra. *A Discipline of Programming.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1976.

[18] M. Dumas, W. M. van der Aalst, and A. H. ter Hofstede. *Process-aware Information Systems: Bridging People and Software Through Process Technology.* John Wiley & Sons, Inc., New York, NY, USA, 2005.

[19] M. Eaddy, T. Zimmermann, K. Sherwood, V. Garg, G. Murphy, N. Nagappan, and A. Aho. Do Crosscutting Concerns Cause Defects? *Software Engineering, IEEE Transactions on*, 34(4):497–515, July 2008.

[20] C. A. Ellis. Information control nets: A mathematical model of office information flow. In *Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems*, volume 3670. Boulder, CO, 1979.

[21] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. *SIGMOD Rec.*, 18(2):399–407, June 1989.

[22] C. A. Ellis, S. J. Gibbs, and G. Rein. Groupware: Some issues and experiences. *Commun. ACM*, 34(1):39–58, Jan. 1991.

[23] M. Endrei, J. Ang, A. Arsanjani, S. Chua, P. Comte, P. Krogdahl, M. Luo, and T. Newling. *Patterns: service-oriented architecture and web services*. IBM Corporation, International Technical Support Organization, 2004.

[24] D. C. Engelbart and W. K. English. A research center for augmenting human intellect. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), pages 395–410, New York, NY, USA, 1968. ACM.

[25] T. Erl. Service-oriented architecture. *Concepts, Technology, and Design*, 2004.

[26] T. Erl. *SOA: principles of service design*, volume 1. Prentice Hall Upper Saddle River, 2008.

[27] E. Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.

[28] D. F. Ferraiolo, J. F. Barkley, and D. R. Kuhn. A role-based access control model and reference implementation within a corporate intranet. *ACM Trans. Inf. Syst. Secur.*, 2(1):34–64, Feb. 1999.

[29] D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-Based Access Control*. Artech House, second edition, 2007.

[30] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *27th International Conference on Software Engineering (ICSE 2005), 15-21 May 2005, St. Louis, Missouri, USA*, pages 196–205, 2005.

[31] N. Fraser. Differential synchronization. In *Proceedings of the 9th ACM Symposium on Document Engineering*, DocEng '09, pages 13–20, New York, NY, USA, 2009. ACM.

[32] M. P. Gallaher, A. C. O'Connor, and B. Kropp. The Economic Impact of Role-Based Access Control. Technical report, National Institute of Standards and Technology, 2002.

[33] A. Garcia, C. Sant'Anna, E. Figueiredo, U. Kulesza, C. Lucena, and A. von Staa. Modularizing design patterns with aspects: a quantitative study. In *Transactions on Aspect-Oriented Software Development I*, pages 36–74. Springer, 2006.

[34] P. Gaubatz, W. Hummer, U. Zdun, and M. Strembeck. Supporting Customized Views for Enforcing Access Control Constraints in Real-time Collaborative Web Applications. In *13th International Conference on Web Engineering*, Aalborg, Denmark, July 2013.

[35] P. Gaubatz, W. Hummer, U. Zdun, and M. Strembeck. Enforcing Entailment Constraints in Offline Editing Scenarios for Real-time Collaborative Web Documents. In *29th Symposium On Applied Computing*, Gyeongju, Korea, March 2014.

[36] P. Gaubatz, I. Lytra, and U. Zdun. Automatic Enforcement of Constraints in Real-time Collaborative Architectural Decision Making. *Journal of Systems and Software*, accepted for publication in January 2015.

[37] P. Gaubatz and U. Zdun. UML2 Profile and Model-Driven Approach for Supporting System Integration and Adaptation of Web Data Mashups. In *4th International Workshop on Lightweight Integration on the Web*, Berlin, Germany, July 2012.

[38] P. Gaubatz and U. Zdun. Supporting Entailment Constraints in the Context of Collaborative Web Applications. In *28th Symposium On Applied Computing*, Coimbra, Portugal, March 2013.

[39] C. K. Georgiadis, I. Mavridis, G. Pangalos, and R. K. Thomas. Flexible team-based access control using contexts. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, SACMAT '01, pages 21–27, New York, NY, USA, 2001. ACM.

[40] M. Goldman, G. Little, and R. C. Miller. Real-time collaborative coding in a web ide. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST'11, pages 155–164, New York, NY, USA, 2011. ACM.

[41] G. S. Graham and P. J. Denning. Protection: Principles and practice. In *Proceedings of the May 16-18, 1972, Spring Joint Computer Conference*, AFIPS '72 (Spring), pages 417–429, New York, NY, USA, 1972. ACM.

[42] P. Greenwood, T. Bartolomei, E. Figueiredo, M. Dosea, A. Garcia, N. Cacho, C. Sant'Anna, S. Soares, P. Borba, U. Kulesza, et al. On the impact of aspectual decompositions on design stability: An empirical study. In *ECOOP 2007–Object-Oriented Programming*, pages 176–200. Springer, 2007.

[43] C. A. Gutwin, M. Lippold, and T. C. N. Graham. Real-time groupware in the browser: Testing the performance of web-based networking. In *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work*, CSCW '11, pages 167–176, New York, NY, USA, 2011. ACM.

[44] M. Hafner, R. Breu, B. Agreiter, and A. Nowak. Sectet: an extensible framework for the realization of secure inter-organizational workflows. *Internet Research*, 16(5):491–506, 2006.

[45] A. H. M. T. Hofstede, W. van der Aalst, M. Adams, and N. Russell. *Modern Business Process Automation: YAWL and Its Support Environment*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[46] A. W. Holt. Coordination technology and petri nets. In *Advances in Petri Nets 1985, Covers the 6th European Workshop on Applications and Theory in Petri Nets-selected Papers*, pages 278–296, London, UK, UK, 1986. Springer-Verlag.

[47] W. Hummer, P. Gaubatz, M. Strembeck, U. Zdun, and S. Dustdar. An Integrated Approach for Identity and Access Management in a SOA Context. In *16th ACM Symposium on Access Control Models and Technologies*, Innsbruck, Austria, June 2011.

[48] W. Hummer, P. Gaubatz, M. Strembeck, U. Zdun, and S. Dustdar. Enforcement of Entailment Constraints in Distributed Service-based Business Processes. *Information and Software Technology*, 55(11), November 2013.

[49] W. L. Hürsch and C. V. Lopes. Separation of concerns. 1995.

[50] A. Inthiran and A. Seddon. Security Policies: Making it Work. In *Proceedings of the 6th European Conference on Information Warfare & Security*, page 109. Academic Conferences Limited, 2007.

[51] S. Jablonski. On the complementarity of workflow management and business process modeling. *SIGOIS Bull.*, 16(1):33–38, Aug. 1995.

[52] J. B. D. Joshi, W. G. Aref, A. Ghafoor, and E. H. Spafford. Security Models for Web-based Applications. *Communications of the ACM*, 44(2):38–44, Feb. 2001.

[53] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. *Aspect-oriented programming*. Springer, 1997.

[54] D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA: service-oriented architecture best practices*. Prentice Hall Professional, 2005.

[55] B. W. Lampson. Protection. *SIGOPS Oper. Syst. Rev.*, 8(1):18–24, Jan. 1974.

[56] C. E. Landwehr. Formal models for computer security. *ACM Comput. Surv.*, 13(3):247–278, Sept. 1981.

[57] M. Lippert and C. V. Lopes. A study on exception detection and handling using aspect-oriented programming. In *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, pages 418–427. IEEE, 2000.

[58] I. Lytra, P. Gaubatz, and U. Zdun. Two Controlled Experiments on Model-based Architectural Decision Making. submitted to *Information and Software Technology*, submitted first revision in January 2015.

[59] P. W. Mattessich and B. R. Monsey. *Collaboration: what makes it work. A review of research literature on factors influencing successful collaboration.* Amherst H. Wilder Foundation, 1992.

[60] S. J. Mellor, T. Clark, and T. Futagami. Model-driven development: guest editors' introduction. *IEEE software*, 20(5):14–18, 2003.

[61] S. Minocha and P. G. Thomas. Collaborative learning in a wiki environment: Experiences from a software engineering course. *New Review of Hypermedia and Multimedia*, 13(2):187–209, 2007.

[62] S. Mishra and H. Weistroffer. A Framework for Integrating Sarbanes-Oxley Compliance into the Systems Development Process. *Communications of the Association for Information Systems (CAIS)*, 20(1):712–727, 2007.

[63] S. Mogan and W. Wang. The impact of web 2.0 developments on real-time groupware. In *Proceedings of the 2010 IEEE Second International Conference on Social Computing*, SOCIALCOM '10, pages 534–539, Washington, DC, USA, 2010. IEEE Computer Society.

[64] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr 1989.

[65] M. Nowak and C. Pautasso. Team Situational Awareness and Architectural Decision Making with the Software Architecture Warehouse. In *7th European Conference on Software Architecture*, ECSA'13, pages 146–161, Berlin, Heidelberg, 2013. Springer-Verlag.

[66] OASIS. Web Services Business Process Execution Language. URL: http://docs.oasis-open.org/wsbpel/2.0/OS, 2007.

[67] Object Management Group. OMG Unified Modeling Language (OMG UML): Superstructure (Version 2.2). URL: http://www.omg.org/spec/UML/2.2/, February 2009.

[68] Object Management Group. Business Process Model and Notation (BPMN). URL: http://www.omg.org/spec/BPMN/2.0/, January 2011.

[69] S. Osborn, R. Sandhu, and Q. Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Trans. Inf. Syst. Secur.*, 3(2):85–106, May 2000.

[70] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[71] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3):45–77, Dec. 2007.

[72] D. Pinelle and C. Gutwin. A groupware design framework for loosely coupled workgroups. In *Proceedings of the Ninth Conference on European Conference on Computer Supported Cooperative Work*, ECSCW'05, pages 65–82, New York, NY, USA, 2005. Springer-Verlag New York, Inc.

[73] T. Quirchmayr, P. Gaubatz, M. Strembeck, and U. Zdun. Consistency Checking and Resolution Strategies for Runtime Conflicts Resulting from Changes in Process-related RBAC Models. submitted to *Advances in Verifiably Secure Process-aware Information Systems*, submitted in June 2014.

[74] V. S. Rekha and H. Muccini. A Study on Group Decision-Making in Software Architecture. In *IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 185–194, 2014.

[75] S. Rekha V. and H. Muccini. Suitability of Software Architecture Decision Making Methods for Group Decisions. In *Software Architecture*, volume 8627 of *Lecture Notes in Computer Science*, pages 17–32. Springer International Publishing, 2014.

[76] P. S. Ring and A. H. v. d. Ven. Developmental processes of cooperative interorganizational relationships. *The Academy of Management Review*, 19(1):pp. 90–118, 1994.

[77] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[78] R. Sandhu and Q. Munawer. How to do discretionary access control using roles. In *Proceedings of the Third ACM Workshop on Role-based Access Control*, RBAC '98, pages 47–54, New York, NY, USA, 1998. ACM.

[79] R. Sandhu and P. Samarati. Access control: principle and practice. *Communications Magazine, IEEE*, 32(9):40–48, Sept 1994.

[80] R. S. Sandhu. Lattice-based access control models. *Computer*, 26(11):9–19, Nov. 1993.

[81] B. Selic. The pragmatics of model-driven development. *IEEE software*, 20(5):19–25, 2003.

[82] M. Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2(4):333–360, 1994.

[83] M. Sloman and E. Lupu. Security and management policy specification. *Netwrk. Mag. of Global Internetwkg.*, 16(2):10–19, Mar. 2002.

[84] J. Smith. WPF Apps with the Model-View-ViewModel Design Pattern. *MSDN magazine*, 2009.

[85] S. Sobernig, P. Gaubatz, M. Strembeck, and U. Zdun. Comparing Complexity of API Designs: An Exploratory Experiment on DSL-based Framework Integration. In *10th International Conference on Generative Programming and Component Engineering*, Generative Programming and Component Engineering, Portland, OR, USA, October 2011.

[86] M. Strembeck. A Role Engineering Tool for Role-Based Access Control. In *3rd Symposium on Requirements Engineering for Information Security*, 2005.

[87] M. Strembeck. Scenario-driven Role Engineering. *IEEE Security & Privacy*, 8(1):28–35, January 2010.

[88] M. Strembeck and J. Mendling. Modeling Process-related RBAC Models with Extended UML Activity Models. *Information and Software Technology*, 53(5):456–483, May 2011.

[89] M. Strembeck and G. Neumann. An integrated approach to engineer and enforce context constraints in rbac environments. *ACM Trans. Inf. Syst. Secur.*, 7(3):392–427, Aug. 2004.

[90] R. K. Thomas and R. S. Sandhu. Task-based authorization controls (tbac): A family of models for active and enterprise-oriented autorization management. In *Proceedings of the IFIP TC11 WG11.3 Eleventh International Conference on Database Securty XI: Status and Prospects*, pages 166–181, London, UK, UK, 1998. Chapman & Hall, Ltd.

[91] A. M. Thomson and J. L. Perry. Collaboration processes: Inside the black box. *Public Administration Review*, 66:20–32, 2006.

[92] V. K. Vaishnavi and W. K. JR. *Design Science Research Methods and Patterns: Innovating Information and Communication Technology*. Auerbach, 2007.

[93] W. M. P. van der Aalst. Process-aware information systems: Design, enactment, and analysis. In *Wiley Encyclopedia of Computer Science and Engineering*. 2008.

[94] W. M. P. van der Aalst. Transactions on petri nets and other models of concurrency ii. chapter Process-Aware Information Systems: Lessons to Be Learned from Process Mining, pages 1–26. Springer-Verlag, Berlin, Heidelberg, 2009.

[95] W. M. P. van der Aalst and A. Hofstede. Yawl: Yet another workflow language. *Inf. Syst.*, 30(4):245–275, June 2005.

[96] W. M. P. van der Aalst, A. H. M. T. Hofstede, and M. Weske. Business process management: A survey. In *Proceedings of the 2003 International Conference on Business Process Management*, BPM'03, pages 1–12, Berlin, Heidelberg, 2003. Springer-Verlag.

[97] A. van Deursen, A. Mesbah, B. Cornelissen, A. Zaidman, M. Pinzger, and A. Guzzi. Adinda: A knowledgeable, browser-based ide. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE'10, pages 203–206, New York, NY, USA, 2010. ACM.

[98] J. Wainer, P. Barthelmes, and A. Kumar. W-RBAC - A Workflow Security Model Incorporating Controlled Overriding of Constraints. *International Journal of Cooperative Information Systems*, 12(4):455–485, December 2003.

[99] M. Wenzel, L. Gericke, R. Gumienny, and C. Meinel. Towards cross-platform collaboration - transferring real-time groupware to the browser. In *Proceedings of the 17th International Conference on Computer Supported Cooperative Work in Design*, CSCWD '13, pages 49–54, June 2013.

[100] R. Wonohoesodo and Z. Tari. A role based access control for web services. In *Proceedings of the 2004 IEEE International Conference on Services Computing*, SCC '04, pages 49–56, Washington, DC, USA, 2004. IEEE Computer Society.

[101] X. Xiao, A. Paradkar, S. Thummalapenta, and T. Xie. Automated extraction of security policies from natural-language software documents. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 12:1–12:11, New York, NY, USA, 2012. ACM.

[102] C. Zhang and H.-A. Jacobsen. Quantifying aspects in middleware platforms. In *Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 130–139. ACM, 2003.

[103] H. Zhu and M. Zhou. Role-based collaboration and its kernel mechanisms. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 36(4):578–589, July 2006.

[104] M. D. Zisman. *Representation, Specification and Automation of Office Procedures.* PhD thesis, Wharton School, University of Pennsylvania, 1977.

[105] M. zur Muehlen and M. Indulska. Modeling languages for business processes and business rules: A representational analysis. *Inf. Syst.*, 35(4):379–390, June 2010.

Paper A

An Integrated Approach for Identity and Access Management in a SOA Context

The subsequent paper has been published as follows:

W. Hummer, P. Gaubatz, M. Strembeck, U. Zdun, and S. Dustdar. An Integrated Approach for Identity and Access Management in a SOA Context. In *16th ACM Symposium on Access Control Models and Technologies*, Innsbruck, Austria, June 2011.

In this paper, we devised a model-driven approach including a DSL for the definition of RBAC policies in the context of service-based business processes. We leverage the WS-BPEL (Business Process Execution Language for Web services) extension mechanism to annotate process models with custom RBAC directives. Using our SeCoS (Secure Collaboration in Service-based Architectures) prototype implementation we could demonstrate the feasibility of automatically augmenting an annotated WS-BPEL process with additional tasks concerning the enforcement of the defined RBAC policies. Our approach thus enables (non-technical) domain experts, such as physicians or hospital clerks, to participate in defining and maintaining policies. Thereby, they are able to precisely define the set of users that shall be granted the permission to execute certain tasks within a collaborative business process.

An Integrated Approach for Identity and Access Management in a SOA Context

Waldemar Hummer[1], Patrick Gaubatz[2], Mark Strembeck[3], Uwe Zdun[2], and Schahram Dustdar[1]

[1]Distributed Systems Group
Information Systems Institute
Vienna University of Technology
{lastname}@infosys.tuwien.ac.at

[2]Software Architecture Group
Faculty of Computer Science
University of Vienna
{firstname.lastname}@univie.ac.at

[3]Information Systems Institute
Vienna University of
Economics and Business
mark.strembeck@wu.ac.at

ABSTRACT

In this paper, we present an approach for identity and access management (IAM) in the context of (cross-organizational) service-oriented architectures (SOA). In particular, we defined a domain-specific language (DSL) for role-based access control (RBAC) that allows for the definition of IAM policies for SOAs. For the application in a SOA context, our DSL environment automatically produces WS-BPEL (Business Process Execution Language for Web services) specifications from the RBAC models defined in our DSL. We use the WS-BPEL extension mechanism to annotate parts of the process definition with directives concerning the IAM policies. At deployment time, the WS-BPEL process is instrumented with special activities which are executed at runtime to ensure its compliance to the IAM policies. The algorithm that produces extended WS-BPEL specifications from DSL models is described in detail. Thereby, policies defined via our DSL are automatically mapped to the implementation level of a SOA-based business process. This way, the DSL decouples domain experts' concerns from the technical details of IAM policy specification and enforcement. Our approach thus enables (non-technical) domain experts, such as physicians or hospital clerks, to participate in defining and maintaining IAM policies in a SOA context. Based on a prototype implementation we also discuss several performance aspects of our approach.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access Control*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Client/server, Distributed applications*; D.2.11 [**Software**]: Software Architectures—*Domain-specific architectures, Languages, Service-oriented architecture*

General Terms

Design, Languages, Management, Security

Keywords

Identity and Access Management, SAML, SOAP, WS-BPEL, WS-Security

SACMAT'11, June 15–17, 2011, Innsbruck, Austria.

1. INTRODUCTION

In recent years, Service-Oriented Architectures (SOA) [24] have emerged as a suitable means to develop loosely coupled distributed systems. Today, Web services are a commonly used technology that build the foundation of SOAs and both intra- and cross-organizational business processes. Electronic business collaborations require enforcement of high-level security constraints such as ensuring the identity and competencies of end users, restricted access to resources, or protection of private data. In our previous work, we identified the need for modeling support of identity and access control models from the experiences gained in the area of role engineering (see, e.g., [34–37]). However, to enforce the corresponding access control policies in a software system, the resulting models must also be mapped to the implementation level.

Different aspects of identity and access management (IAM) in distributed environments and SOAs have been studied previously. In fact, our work builds on a number of existing approaches and standards. An important point with regards to electronic business processes spanning multiple services and cross-organizational units is the concept of Single Sign-On (SSO, e.g., [17, 25]), which simplifies user authentication for the individual services by establishing trust relationships across security domains. SSO allows the business process to obtain a signed authentication token for a security domain d, which is also accepted by other security domains that trust domain d. The Security Assertion Markup Language (SAML) [20] provides a standard way of expressing signed assertions about the identity and attributes of a system participant. The Web Services Security (WS-Security) [21] SAML Token Profile defines how SAML assertions can be transported securely in Web service invocations, i.e., by including a security token element in the header of the SOAP (Simple Object Access Protocol) invocation message.

Cross-organizational IAM involves stakeholders with different background and expertise. The technical IAM model which expresses well-defined semantics and supports detailed security audits may be suited for software architects and developers, but for non-technical domain experts an abstracted view is desirable. In the context of model-driven development (MDD) [28,29,33], a systematic approach for DSL (*domain-specific language*) development has emerged in recent years (see, e.g., [15, 32, 38, 42]). A DSL is a tailor-made (computer) language for a specific problem domain. In general, DSLs provide relevant domain abstractions as first class language elements and can be designed and used on different abstraction layers, ranging from DSLs for technical tasks to DSLs for tasks on the business-level. Thus, DSLs can also be defined for non-technical stakeholders, such as business analysts or biologists, for example. In general, a DSL makes domain-knowledge explicit.

That is, the DSL is built so that domain experts can understand and modify DSL code to phrase domain-specific statements that are understood by an information system. To ensure compliance between models and software platforms, the models defined in a DSL are mapped to source code artifacts of the software platform via automated model-transformations (see, e.g., [14, 30, 41]).

This paper presents an approach to define and enforce IAM policies in cross-organizational SOA business processes. The approach is based on the Web Services Business Process Execution Language (WS-BPEL) [22], which has in the previous years emerged as the de-facto standard for defining Web service compositions and business processes. WS-BPEL is an XML-based special-purpose language whose features range from invocation of external Web services, message correlation and asynchronous invocations to control flow structures (e.g., loops, branches, parallel flows), XML data transformation and modification of SOAP message headers. Our implementation builds on well-established standards including SAML and WS-Security, and supports the concept of single-sign-on (SSO) to authorize and secure the individual steps in the business process. The use of a domain-specific language (DSL) for Role Based Access Control (RBAC) [4, 5, 27] allows us to abstract from technological details and to involve domain experts in the security modeling process. SOA experts and software developers utilize the identity and access models to define security constraints while designing electronic business processes in WS-BPEL. At deployment time, the WS-BPEL process is instrumented with special activities to ensure its compliance to the IAM policies at runtime.

The remainder of this paper is structured as follows. In Section 2, we introduce an illustrative scenario for IAM in a distributed SOA context. We then present in Section 3 our approach for integrated modeling and enforcement of identity and access control in SOA business processes, and discuss the mapping from the modeling to the implementation level. Details on the implementation are given in Section 4, and in Section 5 we evaluate different aspects of our solution. Section 6 contains a discussion of related work, and Section 7 concludes the paper with an outlook for future work.

2. SCENARIO: IAM IN A SOA BUSINESS PROCESS CONTEXT

We illustrate the concepts of this paper based on a motivating scenario taken from the e-health domain. Our example scenario models the workflow of an orthopedic hospital which treats fractures and other serious injuries. The hospital is supported by an IT infrastructure organized in a SOA, implemented using Web services. The SOA provides services for patient data, connects the departments of the hospital and facilitates the routine processes. The hospital exchanges data with other partner hospitals. As patient data constitute sensitive information, security must be ensured and a tailored domain-specific RBAC model needs to be enforced.

A core procedure in the hospital is the patient examination. The corresponding technical business process is depicted in Business Process Modeling Notation (BPMN) in Figure 1. We assume that the process is implemented using WS-BPEL and that each BPMN service task (depicted as gray rounded rectangles) denotes the invocation of a Web service. The arrows between the tasks indicate the control flow of the process. The BPMN groups in the figure are annotated with *Role* and *Context* labels, the purpose of which will be detailed later in this section. Note that all tasks are backed by Web services, however, part of the tasks are not purely technical but involve some sort of human labor or interaction. For instance, the activation of the task *Obtain X-Ray Image* triggers an invocation

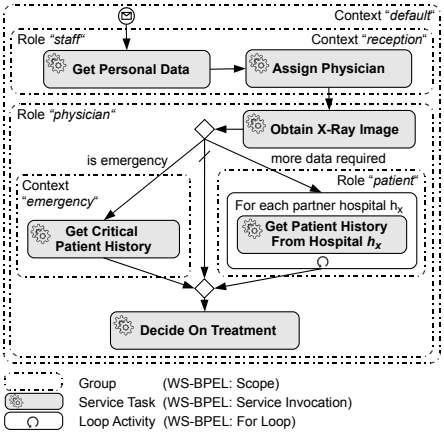


Figure 1: Hospital Patient Examination Scenario in BPMN

to the Web service `http://h1.com/xray`, but the task itself is performed by the hospital staff (and the patient).

The first step in the examination process is to retrieve the personal data of the patient. To demonstrate the cross-organizational character of this scenario, suppose that the patient has never been treated in our example hospital (H1) before, but has already received medical treatment in a partner hospital (H2). Consequently, H1 obtains the patient's personal data from H2 via a Web service residing under the URL `http://h2.com/patients`. Secondly, the patient is assigned to an available physician, which is performed using an `examination` service. These first two tasks need to be performed by a general staff member (role "staff"). In the process definition in Figure 1, this requirement is expressed as a BPMN *group* (rounded rectangle with dashed border) with a corresponding label. In the implementation of the process, this group is mapped to a BPEL *scope* with an extensibility attribute `role`. Similar to a scope in a regular programming language, a WS-BPEL scope embraces a set of instructions and defines boundaries for the lifetime of variables and event handlers defined in this scope. Analogously, the role attribute is valid within the boundaries of its owner scope.

After the patient has been assigned, the responsible physician requests an x-ray image using the Web service of the x-ray department (`http://h1.com/xray`). This activity runs under a new group (or scope), which requires the role "physician". The physician then analyzes the received x-ray image and decides whether additional data are required. For instance, the patient may have had a similar fraction or injury in the past, in which case special treatment is required. Hence, the business process requests historical data from partner hospitals, which also participate in the SOA. Due to privacy issues, the historical data are only disclosed to the patient herself, and the *Get Patient History* service task executes under the role "patient". Note that this role change and the identity management is enforced by the platform, which will be discussed in Section 3. Another situation that requires additional data is the case of an emergency. If the emergency demands for immediate surgery, it is important to determine historical data about any critical conditions or diseases that might interfere with the surgery. This critical information is stored in a secured repository which can be accessed via the Web service `http://h1.com/emergency`. Access to the critical historical data requires the context "emergency", which

is also indicated via an enclosing scope in Figure 1. Finally, after acquiring the necessary data, the process switches back to the context "default" and the role "physician". The invocation of the operation `decideOnTreatment` constitutes the end of the examination and triggers the subsequent treatment activities.

The following list summarizes the stakeholders and their key requirements concerning the SOA-based IT system of the hospital.

- The IT system facilitates the hospital *staff* in their daily work and employs a clear role concept for separation of concerns.
- Besides receiving an efficient treatment, the main interest of the *patient* is that all personal data remain confidential and protected from abuse.
- The *security experts* of the hospital need not necessarily be technical experts and hence require an intuitive interface to model identities, roles and security restrictions in the system.
- The IT *architects and developers* who implement Web services and business processes desire an integrated solution, in which identity and access control can be easily plugged in based on the models defined by the hospital's management.

In the course of this paper, we focus on two aspects concerned with mapping security constraints from a higher-level model to the implementation level: 1) enabling domain experts to map the identity and access model from its abstract representation to a DSL, and mapping of DSL expressions to the implementation level, 2) enabling architects and developers to easily author SOA business processes in WS-BPEL which enforce the security constraints.

3. INTEGRATED APPROACH FOR IAM IN A SOA CONTEXT

This section presents our integrated approach for identity and access management and enforcement in a SOA context. The core assets in a SOA are the services, and the participants that perform operations on these services are either humans or other services. It has been shown that SOA models can be mapped to (extended) RBAC models (e.g., [1]). We build on these findings and provide a declarative DSL for RBAC, integrated with an end-to-end solution for simplified development of secured SOA business processes. The tight integration of the DSL allows to trace identity and access control specifications from the modeling level down to the implementation code, enabling the detailed audit of security compliance.

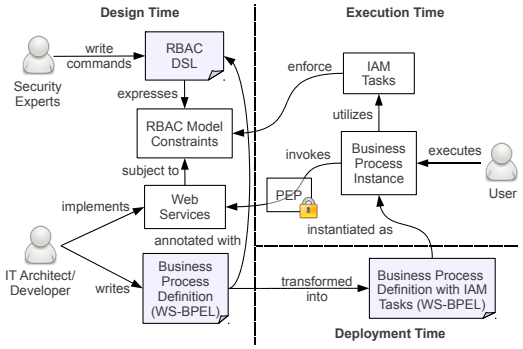


Figure 2: Approach Overview

Figure 2 depicts a high-level overview of our approach, including the involved stakeholders and system artifacts and the relationships between them. At design time, the security experts write RBAC DSL commands to define the RBAC model constraints. The IT specialists implement Web services and define WS-BPEL processes on top of the services. The WS-BPEL definition is annotated with elements from the RBAC DSL, in order to define which parts in the process require which access privileges. At deployment time, the WS-BPEL file is automatically enriched with IAM tasks that conform to the security annotations. The business process is instantiated and executed by human individuals (for example patients and staff members), and the IAM tasks have the process conform to the constraints defined in the RBAC model. A PEP component intercepts all service invocations and blocks unauthorized access.

In the following, we firstly discuss the core language model of the RBAC DSL and show its mapping to the textual representation and further down to the implementation level. Secondly, we present our approach for automatic enforcement of the access control constraints using the extensibility mechanism in WS-BPEL processes.

3.1 DSL-Based RBAC Modeling for SOA

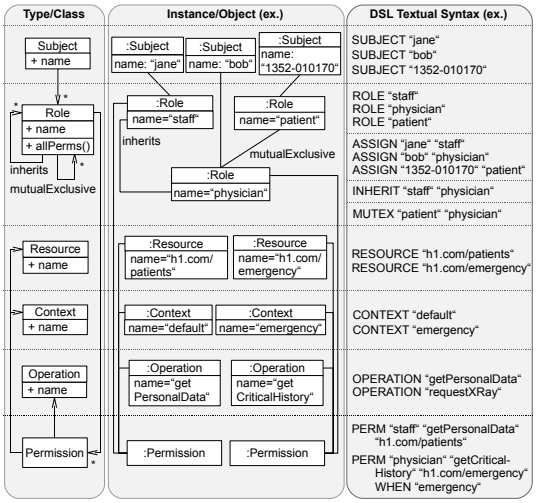


Figure 3: RBAC Model and DSL Language Elements

Figure 3 depicts an example that shows the different abstraction layers of our RBAC DSL. In particular it depicts a (simplified) class diagram of the DSL language elements, an excerpt of the object diagram for the hospital scenario from Figure 1, and a textual representation of the example specified with our RBAC DSL. `Subjects` are identified by a `name` attribute: hospital staff receive a unique name, and for the patients' name we use their social security number, which serves as a unique identifier. Subjects are associated with an arbitrary number of `Roles`, which are themselves associated with `Permissions` to execute certain `Operations`. Roles may inherit from other role instances (association `inherits`), and two roles can be defined as being mutually exclusive (association `mutualExclusive`). We use a context-specific extension of the traditional RBAC model, which has been proposed previously in a similar form (e.g., [6, 8, 26]). The `Context` element allows for a more fine-grained definition of permissions and maps directly to the context requirements in the scenario process definition (see Figure 1). In our approach, we directly associate Web service instances with `Resources`, service invocations with RBAC `Operations`, and `Contexts` with scopes in a Web services business process. A scope in WS-BPEL builds a group of related tasks and limits the lifetime and validness of its enclosed variables, partner links, correlation sets and event handlers. The RBAC permis-

sions are expressed with regard to a certain context in which they are applicable. When a WS-BPEL scope is associated with a certain context (e.g., *emergency*), then all activities (i.e., operations) contained in that scope must execute under this context, and, consequently, the subject executing the process must be allowed to invoke the service operations under this context (see Section 3.2). For instance, when the physician named *bob* is about to retrieve the critical patient history in our scenario, then *bob* needs to have the role *physician*, which allows him to execute the Web service operation `getCriticalHistory` in the context *emergency* (see Figure 3). The *default* context always exists and is automatically assumed if no context is explicitly provided.

DSL Command	Effect (OCL)
`SUBJECT "jane"`	Subject.allInstances()->select(s | s.name='jane')->size() = 1
`ASSIGN "jane" "staff"`	Subject.allInstances()->select(s | s.name='jane').role->select(r | r.name='staff')->size() = 1
`INHERIT "staff" "physician"`	Role.allInstances()->select(r1 | r1.name='staff').allPerms()->forAll(p1 | Role.allInstances()->select(r2 | r2.name='physician').allPerms() ->exists(p2 | p1=p2))
`MUTEX "patient" "physician"`	Subject.allInstances()->forAll(s | not (s.role->exists(r | r.name='physician') and s.role->exists(r | r.name='patient')))

Table 1: Excerpt of RBAC DSL Semantics in OCL

An excerpt of the RBAC DSL constructs and their effect expressed as an OCL (Object Constraint Language) expression is printed in Table 1. The first exemplary command, `SUBJECT "jane"` has the effect that, upon execution, exactly one instance of the class *Subject* with `name` attribute *"jane"* exists. The effect of the second instruction is that the *Subject* named *jane* has an associated *Role* object with name *"staff"*. The `INHERIT` command takes two parameters, a junior-role and a senior-role name, and causes the senior-role to inherit all permissions of the junior-role. The operation *Role.allPerms()* returns all associated permissions of a Role instance and its ancestor roles. Finally, the statically mutual exclusive roles *"patient"* and *"physician"* are defined via the DSL command `MUTEX`, which specifies that no *Subject* instance must ever be assigned both of these roles simultaneously . We currently do not use the alternative form of dynamic mutually exclusive constraints which disallow combinations of certain roles to be activated by one user in the same session or process instantiation, but this is planned for future work. The four OCL constraints illustrate the mapping from the abstract RBAC domain model to the level of an intermediate language (DSL), which is easy to use and comprehend for domain experts, and abstracts from the underlying complexity. The remaining OCL constraints for our example have been left out for brevity.

3.1.1 Collaborative Identity and Access Modeling for Single-Sign On

The goal of the patient examination scenario is that hospitals are able to *collaboratively* model the identity and access control information. To avoid a single point of failure and because each hospital reserves the right to define their own (internal) access control policies, the RBAC information is not stored centrally, but each hospital maintains their own model. However, the ability to retrieve the model data from partner hospitals is vital in order to support SSO and cross-organizational access to resources. For instance, the loop in the business process in Figure 1 retrieves the patient history from partner hospitals using a secured Web service operation `getPatientHistory`, which is provided by all hospitals. The idea is to store data in a decentralized manner, i.e., when a patient is registered or examined in hospital X, then X creates a patient record that is stored locally, but can be accessed by the partner hospitals. The invocation of the `getPatientHistory` operation is secured with a SAML header asserting the identity of the patient. Consider the patient is identified under a subject name "1352-010170" (cf. Figure 3). This requires that the RBAC models of the partner hospitals also contain a subject with this identifier, and that this subject is associated with the role "patient".

To achieve an integrated view on a distributed RBAC model, different strategies have been proposed. The special-purpose language PCL (Policy Combining Language) defined in [10] allows combining of access control policies expressed in XACML. In another work, integration of policies from different organizations is performed based on the similarity of XACML rules [12]. Since the RBAC DSL essentially provides a subset of the functionality of XACML, we are able to utilize these existing solutions for policy integration and collaborative modeling of access control constraints across the different hospitals in the scenario.

3.2 Security Enforcement in WS-BPEL Processes using Annotations

Section 3.1 discussed how the RBAC model is constructed by means of the RBAC DSL, and how the access constraints relate to services, operations, and scopes in SOA business processes. To enforce these constraints at runtime, the business process needs to follow a special procedure. For instance, invoking the `getPersonalData` operation of Hospital 1 requires the process to execute under the role "staff". That is, this service operation requires the presence of a corresponding SAML WS-Security token in the SOAP header of the request. The token contains a SAML assertion that confirms the identity of the subject executing the process operation, as well as the attribute claims for that subject. Integrity of this token and the contained attributes is ensured by applying an XML signature [40] using the X.509 certificate issued for Hospital 1. The attribute claims contain the information under which *role* ("staff") and in which *context* ("default") the subject executes the operation. To obtain the signed SAML assertion, the process needs to invoke the operation `requestSAMLAssertion` of the SAML Identity Provider (IdP) service of Hospital 1. The patient data service relies on the IdP to identify and authenticate the subject (process user), hence the user credentials (e.g. subject name and password) are required for invocation of `requestSAMLAssertion`.

Since one execution of the patient examination process involves different subjects (a staff member, a physician, a patient), the user credentials cannot be hard-coded into the process definition, but are requested from a separate, decoupled *Credentials Provider* (CrP) service. This service offers a `getUserAuthentication` operation, which provides the actual user credentials to be used for a specific process scope. Upon invocation, this operation will cause a username/password input prompt to be displayed to the staff member sitting at the reception desk. After the user has been authenticated, the user credentials can also be stored in a local session configuration file on the reception desk computer. To avoid plaintext passwords from being transmitted over the network, the returned user credentials are encrypted using WS-Security [21]. During execution, the *Credentials Provider* service is always invoked when the process enters a scope that requires a change of subject.

The detailed procedure is illustrated in Figure 4, which shows the sub-part of the hospital scenario process that executes under

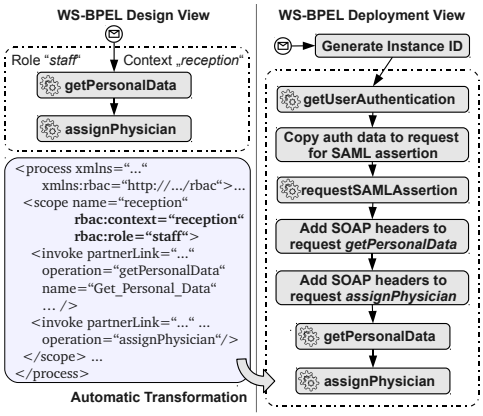


Figure 4: Transformation of WS-BPEL Process Definition

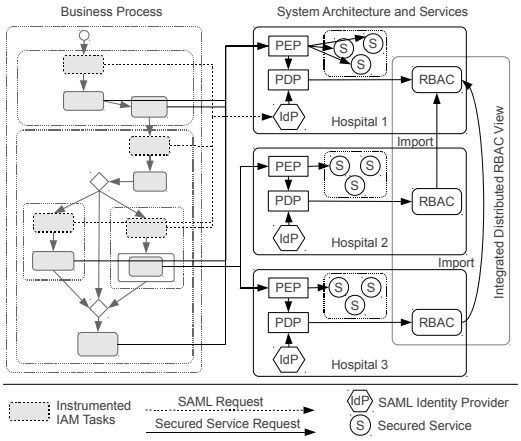


Figure 5: Example Process in System Architecture

the role "staff" and the context "reception". The left part of the figure shows the process definition at design time. Note the annotation attributes `rbac:context` and `rbac:role` which define the required context and role for the scope. At deployment time, the necessary additional process tasks are inserted into the WS-BPEL definition by means of an automatic transformation. At the start of the transformed process, an activity is inserted which generates a unique process instance identifier (ID). The instance ID is sent along as a SOAP header in all subsequent invocations of the WS-BPEL process. This ID helps the CrP service to correlate previous invocations of the process instance, and to keep track of the process state in order to provide the credentials from the correct subject. For instance, when the CrP's operation `getUserAuthentication` is first called with the generated ID, the user credentials are requested from the reception desk employee. The second invocation with the same instance ID will cause the CrP to request the user credentials from the assigned physician, and so on (cf. Figure 1). Note that the CrP service is application-specific and constitutes a tailor-made decoupled component that orchestrates the retrieval of user credentials of changing subjects. The injected process tasks that follow the CrP invocation retrieve the required SAML assertion from the IdP and copy a corresponding SAML header to all service requests of the scope. Details on the implementation of the automatic WS-BPEL transformation are provided in Section 4.

4. IMPLEMENTATION

In the following, we describe the prototype implementation of our approach for integrated SOA identity and access control. This section is divided into four parts: firstly, we outline the architecture of the system and the relationship between the individual services and components; secondly, the SAML-based SSO mechanism is described; the third part briefly discusses the implementation of the RBAC DSL; finally we present the algorithm for automatic transformation of WS-BPEL definitions containing security annotations.

4.1 System Architecture

Figure 5 sketches the high-level architecture and relationship between the example process and the system components. The patient examination example scenario is implemented using WS-BPEL [22] and deployed in a Glassfish[1] server with WS-BPEL module. The example scenario involves three hospitals, which host the protected services for patient management and examination. All service invocations are routed through a Policy Enforcement Point (PEP), which acts as a central security gateway, intercepts every incoming service request and either allows or disallows its invocation. Using the Java API for XML Web services (JAX-WS), the PEP has been implemented as a SOAP message handler (interface `SOAPHandler`). This handler can be plugged into the Web service's runtime engine in a straightforward manner. Once activated, the interceptor is able to inspect and modify inbound and outbound SOAP messages as well as to abort the service invocation.

Each hospital runs an instance of the SAML IdP service, which is used to issue the SAML assertions that are required in the WS-BPEL process. The responsibilities of the IdP are twofold: firstly, it checks whether the subject (i.e., the user currently executing the process) has provided valid credentials; secondly, the IdP assures the identity of a subject and its associated attributes (roles, contexts) by issuing an SAML assertion which is used as a SOAP header in subsequent service invocations by this subject (i.e., the process scope for which it is valid).

The actual decision whether an invocation should be prevented or not is typically delegated to another entity, the Policy Decision Point (PDP). When deciding over the access to a service resource the PDP has to make sure that the subject attempting to access the resource has the permission to do so. In our concrete implementation, the PDP uses the RBAC repository to determine whether the requesting subject is permitted to access the target resource (service) under the specified context and role. Thereby, the PDP can rely on the SAML tokens in the SOAP header of the request messages, which assert the identity of the subject as well as the context and role it operates under. The policy information in the RBAC repository is based on the DSL commands authored by domain experts. Each repository defines both local rules and integrates rules from RBAC repositories of trusted partner hospitals (see, e.g., [10, 12]). The combined information of all RBAC repositories creates an integrated view on the distributed RBAC model.

The advantage of our approach is that changing security requirements in the course of the process execution are handled automati-

[1] https://glassfish.dev.java.net/

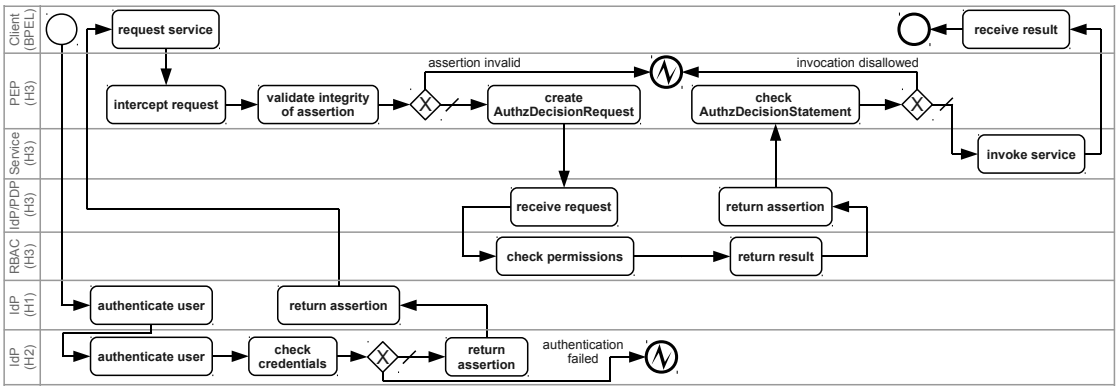


Figure 6: Identity and Access Control Enforcement Procedure

cally. Each time the process changes the scope and requires a new role or context, we utilize the *Instrumented IAM Tasks* which get injected into the WS-BPEL process automatically, as described in Section 3.2. The IAM tasks invoke the IdP and request a new security assertion token for the current subject, role, and context. The security token is then added to the header of all invocations in the same scope. This procedure is repeated for all sub-scopes which require a new role or context. More details concerning the automatic generation of the IAM tasks in WS-BPEL are given in Section 4.4.

4.2 SAML-based Single Sign-On

Figure 6 depicts an example of the Identity and Access Control enforcement procedure modeled via BPMN. To illustrate the SSO aspect of the scenario, we assume that a patient with subject name "1352-010170" (cf. Figure 3), who is registered in hospital 2 (H2), is examined in hospital 1 (H1) and requests its patient history from previous examinations in hospital 3 (H3). The procedure is initiated by the Web service client that demands the execution of a protected Web service. Note that we use the generic term *client*, whereas in our scenario this client is the WS-BPEL engine executing the patient examination process. Prior to issuing the actual service request, the client has to authenticate using the SAML IdP. The latter queries the user database (DB) to validate the credentials provided by the client. In our approach, the credentials (e.g., username-password combinations) are stored in a separate DB and are hence decoupled from the RBAC model. However, the username in the DB equals the subject name in the RBAC model. As the credentials of user "1352-010170" are not stored in the DB of H1, the IdP contacts the IdP of H2, which validates the credentials.

If the user credentials could not be validated, the process is terminated prematurely and a SOAP fault message is returned. In our example scenario, the business process receives the fault message and activates corresponding WS-BPEL fault handlers. Otherwise, if the credentials are valid, the IdP creates a signed assertion similar to the one shown in Listing 1 and passes it back to the client. From now on the business process attaches this assertion to every service request. The request to the protected service is then intercepted by the PEP of H3, which extracts the attached assertion, validates its integrity, and aborts the service invocation if the assertion is invalid (i.e., has been manipulated). Otherwise, it generates an Authorization Decision Request message which is passed to the PDP. The PDP then asks the RBAC repository if the client is allowed to access the requested service. The PDP's decision is expressed as an Authorization Decision Statement. Wrapped into an assertion similar to the one shown in Listing 2, the statement is then passed back to the PEP. Based on the assertion's enclosed information the PEP then either effects the actual service invocation or returns a fault.

The example SAML assertion in Listing 1 illustrates the information that is encapsulated in the header token when the scenario process invokes the `getPatientHistory` operation of the patient Web service of H3. The assertion states that the subject named `1352-010170`, which has been successfully authenticated by the IdP of the hospital denoted by the `Issuer` element (H2), is allowed to use the context `default` and the role `patient`. Note that the subject can be a human being, but it may as well be a service itself that attempts to invoke another service as part of a service composition. The included XML signature element ensures the integrity of the assertion, i.e., that the assertion content indeed originates from the issuing IdP (H2) and has not been modified in any way. When the PEP of H3 intercepts the service invocation with the SAML SOAP header, its first task is to verify the integrity of the assertion. The signature verification requires the public key of the IdP that signed the assertion; this key is directly requested from the corresponding IdP (under `http://h2.com/IdP`) using SAML Metadata [19].

```
1  <Issuer>http://h2.com/IdP</Issuer>
2  <ds:Signature>...</ds:Signature>
3  <Subject><NameID>1352-010170</NameID></Subject>
4  <Conditions NotBefore="2010-12-17T09:48:36.171Z"
5              NotOnOrAfter="2010-12-17T10:00:36.171Z"/>
6  <AttributeStatement>
7    <Attribute Name="context">
8      <AttributeValue>default</AttributeValue>
9    </Attribute>
10   <Attribute Name="role">
11     <AttributeValue>patient</AttributeValue>
12   </Attribute>
13 </AttributeStatement>
14 </Assertion>
```

Listing 1: SAML Assertion Example (1)

After the PEP of H3 has verified the message integrity (and thereby authenticated the subject), it needs to determine whether the subject is authorized to access the requested service operation. This is achieved by the PDP service of H3 that allows the PEP to post an SAML Authorization Decision Query. The PDP answers this query by returning an assertion containing at least one SAML Authorization Decision Statement. Using these Decision State-

ments the PDP is able to express the RBAC service's authorization decision using "plain" SAML. Listing 2 shows an example SAML assertion which informs the PEP that our patient is allowed to invoke the action (operation) `getPersonalData` of the resource (Web service) `http://h1.com/patient`. The `Issuer` name of the PDP is the same as for the IdP (`http://h3.com/IdP`).

```
<Assertion>
  <Issuer>http://h3.com/IdP</Issuer>
  <ds:Signature>...</ds:Signature>
  <Subject>
    <NameID>1352-010170</NameID>
  </Subject>
  <AuthzDecisionStatement Decision="Permit"
    Resource="http://h3.com/patient">
    <Action>getPersonalData</Action>
  </AuthzDecisionStatement>
</Assertion>
```

Listing 2: SAML Assertion Example (2)

4.3 RBAC DSL Implementation

In Section 3.1 we have described the mapping of the RBAC model elements to the textual DSL representation. The mapping of the RBAC DSL commands to executable code on the implementation level is illustrated in Figure 7. We follow a *hybrid* approach to DSL development, which combines *preprocessing* with *embedding* [42]. Embedding means that the DSL platform makes use of an existing *host language* and uses the interpreter and development tools of that language. Preprocessing denotes the process of converting the DSL commands into the machine-readable syntax of the host language using (light-weight) transformations. We evaluated the performance and syntactical flexibility of different (scripting) languages and have chosen Ruby, a language frequently used for DSL development. Ruby provides an interpreter named *JRuby*[2], which is implemented in pure Java and can be integrated using the Java Bean Scripting Framework[3] (BSF).

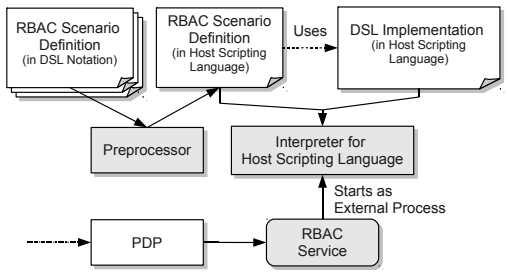


Figure 7: Execution of RBAC Requests

The Preprocessor component transforms the RBAC scenario definition from the DSL notation to the syntax of the host scripting language. An example of a light-weight transformation is to convert `ASSIGN "jane" "staff"` to `ASSIGN "jane","staff"` when using Ruby as the host scripting language. While the first command cannot be parsed by JRuby, the transformed command is well-formed for the interpreter, and is interpreted as a call to the function `ASSIGN` with the two string parameters, which looks up the `Subject` instance and assigns the given role. The remaining DSL constructs are interpreted analogously, and the implementation ensures that all constraints (e.g., mutually exclusive roles, see Table 1) are fulfilled. The preprocessor also serves a second purpose, namely checking whether the DSL code conforms to the allowed syntax or uses any disallowed commands; since the textual DSL is the user interface to the security-critical RBAC model, it is important to identify potentially harmful commands. Another point to consider is that the host language potentially provides features that are undesired for use in the DSL context, such as input/output operations. Hence, the interpreter for the host scripting language executes in a separate Java process, for which we apply restrictive permissions in the Java security policy settings, such as file system access (`java.io.FilePermission`) or network access (`java.net.NetPermission`).

[2]http://jruby.org/

[3]http://jakarta.apache.org/bsf/

4.4 Automatic Transformation of WS-BPEL Process Definition

At deployment time of the business process, the WS-BPEL definition is automatically transformed to ensure correct execution of identity and access control at runtime. Note that the WS-BPEL process is responsible for the choreography of CrP service, SAML IdP, as well as the core business logic services for patient examination.

Figure 8 depicts the relationships between the five scopes (s_1, s_2, s_3, s_4, s_5) of the scenario process. A hierarchical relationship indicates that the child scope (arrow target) is contained in the parent scope (arrow source). Attributes that are not defined in a child scope are inherited from the parent scope. For instance, scope s_3 inherits the context from its parent s_1. The existence of a sequential relationship between two scopes s_x (arrow source) and s_y (arrow target) means that the control flow is passed from s_x to s_y. More specifically, the last task of scope s_x has a control flow link to the first task of s_y in the process definition. This is the case for the scopes s_2 and s_3, where task *Assign Physician* has a control flow link to *Obtain X-Ray Image* (cf. Figure 1). The scope relationships graph is the basis for determining at which points in the process definition the IAM tasks need to be injected.

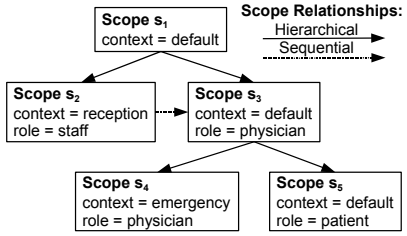


Figure 8: Scope Relationships in Scenario Process

The automatic WS-BPEL transformation is described in Algorithm 1. Variable names are printed in *italics*, and XML markup and XPath expressions are in `typewriter` font. The input is a WS-BPEL document *bpel* with security annotations. Firstly, four required documents need to be imported into the WS-BPEL process using `import` statements: the XML Schema Definitions (XSD) of SAML and WS-Security, and the WSDL (Web Service Description Language) files describing the CrP service and the IdP service.

Then the `partnerLink` declarations for these two services are added to *bpel*, and six `variable` declarations are created (input/output variables for operations `getUserAuthentication` and `requestSAMLAssertion`, a variable to store the assertion, and a variable for additional information such as the instance ID). Next, the algorithm loops over all `scope` elements s with a `role` or `context` attribute, and stores hierarchical and sequential relationships to the array variable *rel*. Although the implementa-

Algorithm 1 WS-BPEL Transformation Algorithm

1: **Input:** WS-BPEL document *bpel*
2: **Output:** transformed BPEL document
3: add `<import ../>` statements to *bpel*
4: add `<partnerLink ../>` definitions to *bpel*
5: add `<variable ../>` declarations to *bpel*
6: $rel \leftarrow$ new array // use variable rel for scope relationships
7: **for all** *bpel*`//scope` as s **do**
8: $rel[s] \leftarrow \emptyset$
9: **if** s`/@role` or s`/@context` **then**
10: $rel[s] \leftarrow rel[s] \bigcup s$`/ancestor::scope[1]`
11: $rel[s] \leftarrow rel[s] \bigcup s$`/preceding-sibling::scope[1]`
12: **end if**
13: **end for**
14: **for all** indexes s in rel, r in $rel[s]$ **do**
15: **if** scopes r and s have different security requirements **then**
16: // add IAM tasks to s: `<invoke>` for IdP and CrP services, `<assign>` (SAML SOAP header) for each `<invoke>` in s
17: **end if**
18: **end for**

tion also considers more complex cases, we assume that a sequential relationship exists if s has a preceding XML sibling element (i.e., an element on the same level as s in the element tree, sharing the parent element with s) named `scope`. The parent scope in a hierarchical relationship can be addressed using the XPath `ancestor::scope[1]`. After all scope relationships have been determined, we loop over all related scopes r and s (conforming to the notation in Figure 8, an arrow points from r to s) and check whether the security requirements are different (in terms of different security annotations). If so, the IAM tasks are injected to the beginning of scope s. The IAM tasks consist of two `invokes` for the invocations to CrP and IdP, as well as several `assign` tasks which add the security SOAP headers to the requests of the remaining service invocations in scope s. Note that the first scope in the process always receives the IAM tasks, although it has neither a parent nor a preceding sibling element.

5. DISCUSSION AND EVALUATION

We evaluated various aspects of the presented solution, and the main evaluation results are discussed in the following. The key aspects are the runtime performance of identity and access control enforcement, and the discussed WS-BPEL transformation algorithm.

To evaluate the scalability of the approach we have defined, deployed, and executed ten test processes with the Glassfish WS-BPEL engine. The processes contain an increasing size (1,2,...,10) of scopes that are annotated with `rbac:role` and `rbac:context` attributes. Each scope contains one `<invoke>` task, which invokes one of the Web service operations of the hospital scenario. The average response time of each service is roughly 200 milliseconds. The processes have been deployed in Glassfish, once with enforced security (i.e., annotated with security attributes, automatically transformed at deployment time), and once in an unsecured version. The deployed processes were executed 10 times and we have calculated the average value to minimize the influence of external effects. Figure 9 illustrates different measurements of the process execution time in milliseconds for both the secured and the unsecured version. The secured version incurs a large overhead, which is hardly surprising considering the fact that for each business logic service the process needs to invoke the CrP, IdP and RBAC services, and applies and checks the XML signatures. However, the measured results indicate that the current implementation leaves room for additional optimization.

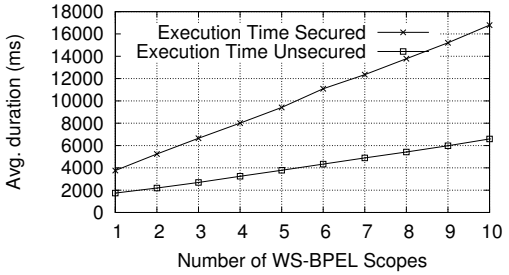


Figure 9: Process Execution Times – Secured vs Unsecured

In Section 4.1 we presented our concrete implementation of an SAML IdP and its duty to issue SAML Assertion tokens. These Assertion tokens are embedded in the header of every subsequent request to secured Web services. Each Assertion contains at least one Attribute Statement that includes the service's required role and context attribute. As the Assertion contains exactly one single context as well as one single role attribute, this means that the Assertion is only valid for one single subject, role, and context. Furthermore this also means, that whenever one of these three change, a new SAML Assertion has to be issued by the IdP. In terms of performance, this approach may not be the most effective, but it has the advantage, that it can be implemented using "plain" SAML. If performance is a critical issue, we propose the following solution: Instead of creating lots of specialized Assertions, the IdP should issue just one generic Assertion per subject. Contrary to the specialized Assertion, the generic one contains a list of all context and role attributes that the subject is allowed to use (instead of just one in each case). This means, that the generic Assertion can be re-used for multiple role/context changes in the WS-BPEL process. Consequently, the IdP's workload can be effectively reduced (provided that there is at least one role/context change present in the WS-BPEL process). The drawback of this solution is that a new custom SOAP header needs to be introduced, in which the client specifies which context and which role (chosen from the Assertion's list of allowed ones) it wants to use. Since the WS-BPEL engine acts as the client, it is the engine's duty to select and attach the correct header to every Web service request. Hence this functionality has to be embedded in the WS-BPEL process definition which, again, increases its size and complexity substantially.

Concerning the evaluation of the WS-BPEL transformation algorithm, we again consider the ten test processes described earlier in this Section. Figure 10 shows the number of WS-BPEL elements of the process definition before and after the automatic transformation. The results indicate that the size of the WS-BPEL definition rises sharply with increasing number of scopes. While our test process with a single scope contains 33/115 WS-BPEL elements before/after transformation, the process definition for 10 scopes grows to 60/484 WS-BPEL elements before/after transformation, respectively. These numbers are determined by counting all XML (sub-)elements in the WS-BPEL file using the XPath expression `count(//*)`. At the beginning of the transformation, 41 elements are added (`import`, `partnerLink` and `variable` declarations), and for each new scope 41 elements are added for the IAM task definitions (note that both values are 41 coincidentally). We observe that the ability to define security annotations in WS-BPEL greatly reduces the required effort at design time.

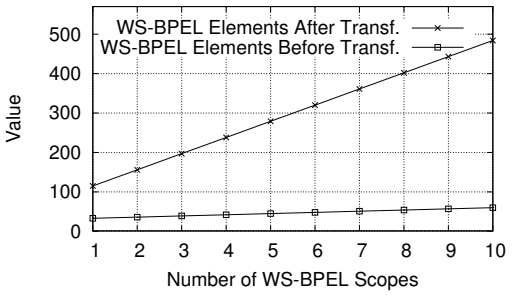


Figure 10: Process Size before and afer Transformation

The textual DSL is used as an interface to the policy rules stored in the RBAC repository. In case a UML binding is required, it is straightforward to integrate our DSL with domain-specific UML extensions for process-related RBAC models (see, e.g., [36]).

6. RELATED WORK

This section discusses related approaches in the area of model-driven IAM and their application to SOA.

Skoksrud et al. present Trust-Serv [31], a solution for model-driven trust negotiation in Web service environments. The platform supports modeling of trust negotiation policies as state machines, and the policy enforcement is transparent to the involved Web services. Different strategies for policy lifecycle management and migration are proposed. Our approach is less concerned with iterative creation of trust relationships, but builds on an IAM model and uses an integrated enforcement in Web service based business processes.

An integrated approach for Model Driven Security, that promotes the use of Model Driven Architectures in the context of access control, is presented by Basin et al. [2]. The foundation is a generic schema that allows creation of DSLs for modeling of access control requirements. The domain expert then defines models of security requirements using these languages. With the help of generators these models are then transformed to access control infrastructures.

The approach by Wolter et al. [39] is concerned with modeling and enforcing security goals in the context of SOA business processes. Similar to our approach, their work suggests that business process experts should collaboratively work on the security policies. A computational independent model (CIM) defines high-level goals, and the CIM gets mapped to a platform independent model (PIM) and further to a platform specific model (PSM). At the PIM level, XACML and *AXIS 2*[4] security configurations are generated. Whereas their approach is more generic and attempts to cover diverse security goals including integrity, availability and audit, we focus on IAM in WS-BPEL business processes.

Kulkarni et al. [9] describe an application of context-aware RBAC to pervasive computing systems. As the paper rightly states, model-level support for revocation of roles and permissions is required to deal with changing context information. Whereas their approach has a strong focus on dynamically changing context (e.g., conditions measured by sensors) and the associated permission (de-)activation, context in our case is a design-time attribute that is part of the RBAC model definitions.

A related access control framework for WS-BPEL is presented by Paci et al. in [23]. It introduces the *RBAC-WS-BPEL* model and the authorization constraint language *BPCL*. Similar to our approach, the BPEL activities are associated with required permissions (in particular, we associate permissions for `invoke` activities that try to call certain service operations). However, one main difference is related to the boundaries of the validity of user permissions: RBAC-WS-BPEL considers pairs of adjacent activities (a_1 and a_2, where a_1 has a control flow link to a_2) and defines rules among them, including separation of duty (a_1 and a_2 must execute under different roles) and binding of duty (a_1 and a_2 require the same role or user); our approach, on the other hand, is to annotate scopes in BPEL processes, which allows to apply separation and binding of duties in a sequential, but also in a hierarchical manner.

A dynamic approach for enforcement of Web services Security is presented in [16] by Mourad et al. The novelty of the approach is mainly grounded by the use of Aspect-Oriented Programming (AOP) in this context, whereby security enforcement activities are specified as *aspects* that are dynamically weaved into the WS-BPEL process at certain *join points*. Essentially, our approach can also be regarded as a variant of AOP: the weaved aspects are the IAM tasks, and join points are defined by security annotations in the process. A major advantage of our approach is the built-in support for SSO and cross-organizational IAM. An interesting extension could be to decouple security annotations from the WS-BPEL definition and to dynamically adapt to changes at runtime.

Various other papers have been published that are related to our work or have influenced it, some of which are mentioned in the following. The platform-independent framework for Security Services named SECTISSIMO has been proposed by Memon at al. [13]. A multilayer mandatory access control (MAC) architecture tailored to Web applications is presented by Hicks et al. [7]. Lin et al. [11] propose policy decomposition to support collaborative access control definition. In [3] an approach to speeding up credential-based access control operations – in particular in the web context – is proposed by Carminati et al.

XACML [18] is an XML-based standard to describe RBAC policies in a flexible and extensible way. Our DSL could be classified as a high-level abstraction that implements a subset of XACML's feature set. Using a transformation of DSL code to XACML markup, it becomes possible to integrate our approach with the well-established XACML environment and tools for policy integration (e.g., [12]).

7. CONCLUSION

We presented an integrated approach for Identity and Access Management in a SOA context. The solution is centered around model-driven development of RBAC constraints, and runtime enforcement of these constraints in Web services based business processes. Our approach fosters cross-organizational authentication and authorization in service-based systems, and greatly simplifies development of SSO-enabled WS-BPEL processes. Although tailor-made SSO solutions (coded explicitly in the business process) may yield a performance gain over the generic approach, from a practical viewpoint our approach has the advantage of being highly reusable and simple to apply. As part of our ongoing work, we are developing alternative ways to define and assign RBAC permissions at runtime, also taking into account dynamic mutual exclusion. We further investigate the use of additional security annotations and an extended view of context information.

8. REFERENCES

[1] M. Alam, M. Hafner, and R. Breu. A constraint based role based access control in the SECTET a model-driven approach. In *Int. Conf. on Privacy, Security and Trust*, 2006.

[4] http://axis.apache.org/axis2/java/core/

[2] D. Basin, J. Doser, and T. Lodderstedt. Model driven security: From UML models to access control infrastructures. *ACM Transactions on Software Engineering Methodology*, 15:39–91, 2006.

[3] B. Carminati and E. Ferrari. AC-XML documents: improving the performance of a web access control module. In *10th ACM SACMAT*, pages 67–76, 2005.

[4] D. F. Ferraiolo and D. R. Kuhn. Role-Based Access Controls. In *15th National Computer Security Conference*, 1992.

[5] D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-Based Access Control*. Artech House, second edition, 2007.

[6] O. Garcia-Morchon and K. Wehrle. Efficient and context-aware access control for pervasive medical sensor networks. In *IEEE Int. Conf. on Pervasive Computing and Communications Workshops*, pages 322 –327, April 2010.

[7] B. Hicks, S. Rueda, D. King, T. Moyer, J. Schiffman, Y. Sreenivasan, P. McDaniel, and T. Jaeger. An architecture for enforcing end-to-end access control over web applications. In *15th ACM SACMAT*, pages 163–172, 2010.

[8] V. Koufi, F. Malamateniou, and G. Vassilacopoulos. A Mediation Framework for the Implementation of Context-Aware Access Control in Pervasive Grid-Based Healthcare Systems. In *4th Int. Conf. on Advances in Grid and Pervasive Computing*, pages 281–292, 2009.

[9] D. Kulkarni and A. Tripathi. Context-aware role-based access control in pervasive computing systems. In *13th ACM SACMAT*, pages 113–122, 2008.

[10] N. Li, Q. Wang, W. Qardaji, E. Bertino, P. Rao, J. Lobo, and D. Lin. Access control policy combining: theory meets practice. In *14th ACM SACMAT*, pages 135–144, 2009.

[11] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo. Policy decomposition for collaborative access control. In *13th ACM SACMAT*, pages 103–112, 2008.

[12] P. Mazzoleni, B. Crispo, S. Sivasubramanian, and E. Bertino. XACML Policy Integration Algorithms. *ACM Transactions on Information System Security*, 11:4:1–4:29, February 2008.

[13] M. Memon, M. Hafner, and R. Breu. SECTISSIMO: A Platform-independent Framework for Security Services. In *Modeling Security Workshop at MODELS '08*, 2008.

[14] T. Mens and P. V. Gorp. A Taxonomy of Model Transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142, 2006.

[15] M. Mernik, J. Heering, and A. Sloane. When and How to Develop Domain-Specific Languages. *ACM Computing Surveys*, 37(4):316–344, December 2005.

[16] A. Mourad, S. Ayoubi, H. Yahyaoui, and H. Otrok. New approach for the dynamic enforcement of Web services security. In *8th Int. Conf. on Privacy Security and Trust*, pages 189 –196, 2010.

[17] B. Neuman and T. Ts'o. Kerberos: an authentication service for computer networks. *Communications Magazine, IEEE*, 32(9):33–38, Sept. 1994.

[18] OASIS. eXtensible Access Control Markup Language. http://docs.oasis-open.org/xacml/2.0, 2005.

[19] OASIS. Metadata for the OASIS Security Assertion Markup Language (SAML). http://docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf, 2005.

[20] OASIS. Security Assertion Markup Language. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf, March 2005.

[21] OASIS. Web Services Security: SOAP Message Security 1.1. http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf, 2006.

[22] OASIS. Web Services Business Process Execution Language. http://docs.oasis-open.org/wsbpel/2.0/OS, 2007.

[23] F. Paci, E. Bertino, and J. Crampton. An Access-Control Framework for WS-BPEL. *Int. J. f. Web Services Research*, 5(3):20–43, 2008.

[24] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, 40(11):38–45, 2007.

[25] A. Pashalidis and C. J. Mitchell. A taxonomy of single sign-on systems. In *8th Australasian Conference on Information Security and Privacy*, pages 249–264, 2003.

[26] W. rong Jih, S. you Cheng, J. Y. jen Hsu, and T. ming Tsai. Context-aware access control in pervasive healthcare. In *EEE Workshop: Mobility, Agents, and Mobile Services*, 2005.

[27] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *Computer*, 29(2):38 –47, 1996.

[28] D. C. Schmidt. Model-Driven Engineering – Guest Editor's Introduction. *Computer*, 39(2), February 2006.

[29] B. Selic. The Pragmatics of Model-Driven Development. *IEEE Software*, 20(5), 2003.

[30] S. Sendall and W. Kozaczynski. Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software*, 20(5), 2003.

[31] H. Skogsrud, B. Benatallah, and F. Casati. Model-Driven Trust Negotiation for Web Services. *IEEE Internet Computing*, 7:45–52, November 2003.

[32] D. Spinellis. Notable design patterns for domain-specific languages. *J. of Systems and Software*, 56(1):91–99, 2001.

[33] T. Stahl and M. Völter. *Model-Driven Software Development*. John Wiley & Sons, 2006.

[34] M. Strembeck. A Role Engineering Tool for Role-Based Access Control,. In *3rd Symposium on Requirements Engineering for Information Security*, 2005.

[35] M. Strembeck. Scenario-driven Role Engineering. *IEEE Security & Privacy*, 8(1), January/February 2010.

[36] M. Strembeck and J. Mendling. Modeling Process-related RBAC Models with Extended UML Activity Models. *Information and Software Technology*, 53(5), May 2011.

[37] M. Strembeck and G. Neumann. An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments. *ACM Trans. on Inf. and System Security*, 7(3), 2004.

[38] M. Strembeck and U. Zdun. An Approach for the Systematic Development of Domain-Specific Languages. *Software: Practice and Experience (SP&E)*, 39(15), October 2009.

[39] C. Wolter, M. Menzel, A. Schaad, P. Miseldine, and C. Meinel. Model-driven business process security requirement specification. *J. Syst. Archit.*, 55:211–223, 2009.

[40] World Wide Web Consortium (W3C). XML Signature Syntax and Processing. http://www.w3.org/TR/xmldsig-core/, 2008.

[41] U. Zdun and M. Strembeck. Modeling Composition in Dynamic Programming Environments with Model Transformations. In *5th Int. Sym. on Software Composition*, 2006.

[42] U. Zdun and M. Strembeck. Reusable Architectural Decisions for DSL Design: Foundational Decisions in DSL Projects. In *14th European Conference on Pattern Languages of Programs (EuroPLoP)*, July 2009.

Paper B

Enforcement of Entailment Constraints in Distributed Service-based Business Processes

The subsequent paper has been published as follows:

W. Hummer, P. Gaubatz, M. Strembeck, U. Zdun, and S. Dustdar. Enforcement of Entailment Constraints in Distributed Service-based Business Processes. *Information and Software Technology*, 55(11), November 2013.

This paper is a substantial extension of the work presented in Paper A. In particular, we complemented our model-driven approach with novel means for for defining and automatically enforcing RBAC-related entailment constraints in distributed service-based business processes. We extended our textual DSL with language abstractions for the specification of entailment constraints. Furthermore, we integrated our approach with the *Business Activity* meta-model [88], a generic approach for the specification of process-related RBAC models including a corresponding UML extension. We have significantly extended our SeCoS prototype implementation and provided an extensive performance evaluation of our approach.

Enforcement of Entailment Constraints in Distributed Service-Based Business Processes

Waldemar Hummer[a], Patrick Gaubatz[b], Mark Strembeck[c], Uwe Zdun[b], Schahram Dustdar[a]

[a] *Distributed Systems Group, Vienna University of Technology, Austria*
[b] *Faculty of Computer Science, University of Vienna, Austria*
[c] *Institute of Information Systems, New Media Lab, Vienna University of Economics and Business, Austria*

Abstract

Context: A distributed business process is executed in a distributed computing environment. The service-oriented architecture (SOA) paradigm is a popular option for the integration of software services and execution of distributed business processes. Entailment constraints, such as mutual exclusion and binding constraints, are important means to control process execution. Mutually exclusive tasks result from the division of powerful rights and responsibilities to prevent fraud and abuse. In contrast, binding constraints define that a subject who performed one task must also perform the corresponding bound task(s). **Objective:** We aim to provide a model-driven approach for the specification and enforcement of task-based entailment constraints in distributed service-based business processes. **Method:** Based on a generic metamodel, we define a domain-specific language (DSL) that maps the different modeling-level artifacts to the implementation-level. The DSL integrates elements from role-based access control (RBAC) with the tasks that are performed in a business process. Process definitions are annotated using the DSL, and our software platform uses automated model transformations to produce executable WS-BPEL specifications which enforce the entailment constraints. We evaluate the impact of constraint enforcement on runtime performance for five selected service-based processes from existing literature. **Results:** Our evaluation demonstrates that the approach correctly enforces task-based entailment constraints at runtime. The performance experiments illustrate that the runtime enforcement operates with an overhead that scales well up to the order of several ten thousand logged invocations. Using our DSL annotations, the user-defined process definition remains declarative and clean of security enforcement code. **Conclusion:** Our approach decouples the concerns of (non-technical) domain experts from technical details of entailment constraint enforcement. The developed framework integrates seamlessly with WS-BPEL and the Web services technology stack. Our prototype implementation shows the feasibility of the approach, and the evaluation points to future work and further performance optimizations.

Keywords: Identity and Access Management, Business Process Management, Entailment Constraints, Service-Oriented Architecture (SOA), WS-BPEL

1. Introduction

The Service-Oriented Architecture (SOA) metaphor has been elaborated by different communities to address different problem areas (such as enterprise application integration or business process management, see, e.g., [1]). Amongst others, it can be seen as a set of technology independent concepts for distributed computing environments. In this context, it has emerged as a popular paradigm for developing loosely coupled distributed systems [2, 3]. Today, Web services [4] are a commonly used technology which serves as a foundation of SOAs, as well as distributed business processes. A distributed business process is an intra-organizational or cross-organizational business process executed in a distributed computing environment (such as SOA). Business processes often require the definition and enforcement of process-related security policies. For example, such requirements result from internal business rules of an organization, or service-level agreements (SLAs) [5] with customers. In addition, numerous regulations and IT standards exist that pose compliance requirements for the corresponding systems. In particular, IT systems must comply with laws and regulations such as the Basel II/III Accords, the International Financial Reporting Standards (IFRS), or the Sarbanes-Oxley Act (SOX). For instance, one important part of SOX compliance is to provide adequate support for definition and enforcement of process-related security policies (see, e.g., [6, 7, 8]).

Role-based access control (RBAC) [9, 10] is a de-facto standard for access control in both research and industry. In the context of RBAC, roles are used to model different job positions and scopes of duty within an information system. These roles are equipped with the permissions to perform their respective tasks. Human users and other active entities (subjects) are assigned to roles according to their work profile [11, 12]. A process-related RBAC model (see, e.g., [13, 14]) enables the definition of permissions and entailment constraints for the tasks that are included in business processes. A *task-based entailment constraint* places some restriction on the subjects who

Email addresses: hummer@infosys.tuwien.ac.at (Waldemar Hummer), patrick.gaubatz@univie.ac.at (Patrick Gaubatz), mark.strembeck@wu.ac.at (Mark Strembeck), uwe.zdun@univie.ac.at (Uwe Zdun), dustdar@infosys.tuwien.ac.at (Schahram Dustdar)

can perform a task *x* given that a certain subject has performed another task *y*. Entailment constraints are an important means to assist the specification and enforcement of compliant business processes (see, e.g., [15, 16, 17, 18, 19, 20]).

Mutual exclusion and binding constraints are typical examples of entailment constraints. Mutual exclusion constraints can be subdivided in *static mutual exclusion* (SME) and *dynamic mutual exclusion* (DME) constraints. A SME constraint defines that two tasks (e.g. Order Supplies and Approve Payment) must never be assigned to the same role and must never be performed by the same subject (to prevent fraud and abuse). This constraint is global with respect to *all process instances* in an information system. In contrast, DME refers to individual process instances and can be enforced by defining that two tasks must never be performed by the same subject in the *same process instance*.

In contrast to mutual exclusion constraints, binding constraints define that two bound tasks must be performed by the *same* entity. In particular, a *subject-binding* constraint defines that the same individual who performed the first task must also perform the bound task(s). Similarly, a *role-binding* constraint defines that bound tasks must be performed by members of the same role but not necessarily by the same individual.

Motivation. As outlined above, entailment constraints are an important means to assist the specification of business processes and control their execution. Yet, the runtime enforcement of entailment constraints in distributed SOA business processes is a complex task, and currently there is still a lack of straightforward solutions to achieve this task. This complexity arises from the fact that the tasks of distributed business processes are performed on independent, loosely coupled nodes in a network. One of the advantages of loosely coupled systems is that the different nodes (i.e. services) can execute their tasks independently of other nodes. However, the enforcement of entailment constraints in a distributed system often requires knowledge that is not available to a single node.

Moreover, to enforce access control policies in a software system, the resulting policy models must also be mapped to the implementation level. To account for different platforms and implementation styles, it is important to first establish the enforcement on a generic and conceptual level, in order to map it to concrete platforms (e.g., SOA, as in our case).

Evidently, enforcement of RBAC policies and constraints has an impact on the execution time of business processes. Depending on the complexity of the constraints and the amount of data that needs to be evaluated, the impact will be more or less severe. While the theory behind RBAC and entailment constraints in business processes has been intensively studied in the past, less attention has been devoted to the runtime enforcement, including performance impacts, of such constraints.

With respect to the rapidly increasing importance of process-aware information systems, the correct and efficient implementation of consistency checks in these systems is an important issue. Therefore, the runtime performance needs to be evaluated thoroughly in order to ensure the efficient execution of business processes that are subject to access constraints.

Approach Synopsis. This paper builds on our previous work from [14, 21]. In [14], we presented a generic approach for the specification of process-related RBAC models including a corresponding UML extension (see also Sections 2 and 3). In [21], we discussed an approach for identity and access management in a SOA context. However, while the enforcement of entailment constraints in a distributed system is a very complex task (see discussion in the Motivation section above), neither [14] nor [21] address this important issue. In this paper, we integrate the approaches from [14, 21] and provide multiple novel contributions. In particular, we present an integrated, model-driven approach for the definition and enforcement of RBAC-related entailment constraints in distributed SOA business processes. We extend our textual DSL from [21] with language primitives for the specification of entailment constraints. Furthermore, we significantly extended our implementation and provide an extensive performance evaluation of our solution.

In general, distributed business processes involve stakeholders with different background and expertise. A technical RBAC model may be well-suited for software architects and developers, but for non-technical domain experts an abstracted view is desirable. In the context of model-driven development (MDD) [22, 23, 24], a systematic approach for DSL (*domain-specific language*) development has emerged in recent years (see, e.g., [25, 26, 27, 28]). A DSL is a tailor-made (computer) language for a specific problem domain. To ensure compliance between models and software platforms, models defined in a DSL are mapped to code artifacts via automated model-transformations (see, e.g., [29, 30, 31]). In our approach, the use of a DSL for RBAC constraints allows us to abstract from technical details and involve domain experts in the security modeling procedure.

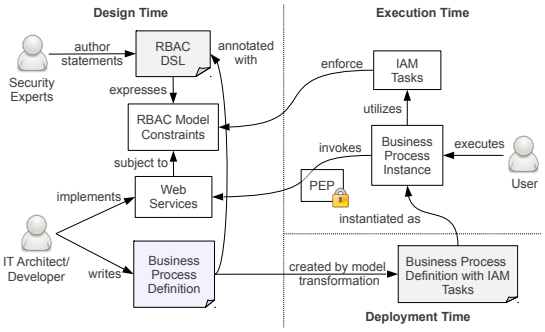


Figure 1: Approach Overview

Figure 1 depicts a high-level overview of our approach, including the involved stakeholders, system artifacts, and relationships between them. At design time, the security experts author RBAC DSL statements to define the RBAC model and entailment constraints. IT specialists implement Web services and define business processes on top of the services. At deployment time, the process definition files are automatically enriched with tasks for identity and access management (IAM) that conform to the corresponding entailment constraints. The

business process is instantiated and executed by human individuals, and the IAM tasks ensure that the process conforms to the constraints defined in the RBAC model. A policy enforcement point (PEP) component intercepts all service invocations to block unauthorized access (see also [21]).

For the sake of platform independence, we model business processes using UML activity diagrams [32]. In particular, we use the *BusinessActivities* extension [14], which enables the definition of process-related RBAC models via extended UML activity models. Based on the generic solution, we discuss a concrete instantiation and show how the approach is mapped to the Web services technology stack, including the Business Process Execution Language for Web services (WS-BPEL) [33].

The remainder of this paper is structured as follows. In Section 2, we present a motivating scenario. Section 3 introduces a generic metamodel for specification of process-related RBAC models including entailment constraints. Section 4 describes the transformation procedure that enriches the process definitions with IAM tasks to enforce runtime-compliance. In Section 5, we present a concrete WS-BPEL-based application of our approach. Implementation-related details are given in Section 6, and in Section 7 we evaluate different aspects of our solution. Section 8 discusses related work, and Section 9 concludes with an outlook for future work.

2. Motivating Scenario

We illustrate the concepts of this paper based on a scenario taken from the e-health domain. The scenario models the workflow of orthopedic hospitals which treat fractures and other serious injuries. The hospitals are supported by an IT infrastructure organized in a SOA, implemented using Web services. The SOA provides Web services for patient data, connects the departments of different hospitals, and facilitates the routine processes. Because the treatment of patients is a critical task and the personal data constitute sensitive information, security must be ensured and a tailored domain-specific RBAC model needs to be enforced. Task-based entailment constraints in the form of mutual exclusion and binding constraints are a crucial part of the system.

2.1. Patient Examination Business Process

A core procedure in the hospital is the patient examination, illustrated in Figure 2 as a *Business Activity* [14] model. We assume that the process is implemented using a business process engine and that the actions (or tasks) represent the invocations of services. The arrows between the actions indicate the control flow of the process. Note that all tasks are backed by technical services, however, part of the tasks are not purely technical but involve some sort of human labor or interaction.

The top part of the figure shows the BusinessActivity model of the process, and the bottom part contains an excerpt of the RBAC definitions that apply to the scenario. We define three types of roles (Staff, Physician, Patient), each with a list of tasks they are permitted to execute, and four subjects (John, Jane, Bob, Alice), each with roles assigned to them. The names of

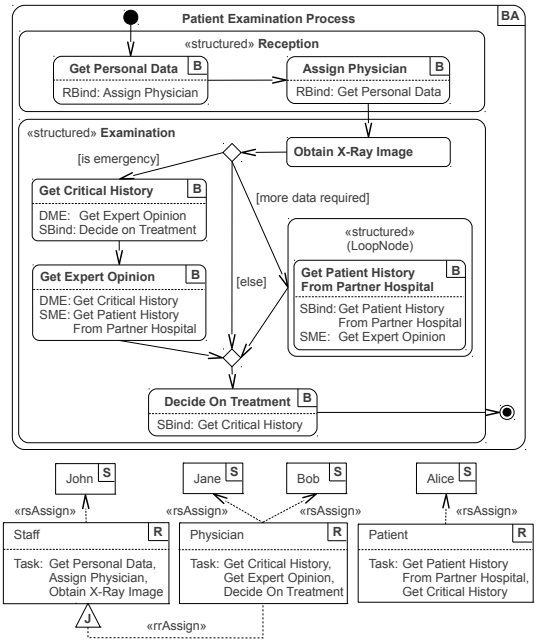


Figure 2: Patient Examination Scenario Modeled As UML Business Activity

permitted tasks of a role are displayed after the string *"Task:"*. Note, however, that this is only one possible graphical presentation option to display the association between roles and actions (see [14]). Role inheritance hierarchies are modeled using the role-to-role assignment (*rrAssign*) relationship (senior-roles inherit the permissions of junior-roles, e.g., Physician inherits from Staff). The role-to-subject assignment (*rsAssign*) association is used to assign roles to subjects.

The first step in the examination process (see Figure 2) is to retrieve the personal data of the patient. To demonstrate the cross-organizational character of this scenario, suppose that the patient has never been treated in our example hospital (H1) before, but has already received medical treatment in a partner hospital (H2). Consequently, H1 obtains the patient's personal data via the Web services of H2. Secondly, the patient is assigned to a physician. After the patient has been assigned, the physician requests an x-ray image from the responsible department. The physician then decides whether additional data are required (e.g., information about similar fractions or injuries in the past). If so, the business process requests historical data from partner hospitals which also participate in the SOA. For privacy reasons, the historical data are only disclosed to the patient herself, and the *Get Patient History* service task has to execute under the role *Patient* (see Figure 2). Another situation that requires additional data is the case of an emergency. If the emergency demands for immediate surgery, it is important to determine historical data about any critical conditions or

diseases that might interfere with the surgery (task *Get Critical History*). To avoid that a single physician takes wrong decisions in an emergency, it is mandatory to get the opinion of a second expert. Finally, the task *Decide On Treatment* completes the examination and triggers the (physical) treatment.

2.2. Entailment Constraints

In this paper, we support four types of entailment constraints which we briefly discuss in the following. The scenario process in Figure 2 contains examples for each type of constraint.

- **Static Mutual Exclusion** (SME): The SME constraint between *Get Expert Opinion* and *Get Patient History from Partner Hospital* defines that the two tasks must never be executed by the same subject or role, across all process instances. This constraint is reasonable as we need to explicitly separate the permissions of patients and physicians.
- **Dynamic Mutual Exclusion** (DME): The DME constraint for *Get Critical History* and *Get Expert Opinion* requires that, for each instance of the process, these two tasks are executed by different subjects. This ensures that the treatment decision in an emergency clearly depends on the medical assessment of two individual physicians.
- **Subject Binding** (SBind): An example SBind constraint is the *Get Patient History From Partner Hospital* task, which executes multiple times in a loop. To ensure that each iteration is done by the same subject, the *SBind* attribute reflexively links to the same task. A second subject binding exists between *Get Critical History* and *Decide on Treatment*.
- **Role Binding** (RBind): The process defines a role-binding constraint which demands that the *Get Personal Data* and *Assign Physician* are performed by the same role (although potentially different subjects).

3. Generic Metamodel for Specification of Entailment Constraints in Business Processes

This section gives an overview of the generic metamodel for specification of process-related RBAC models including entailment constraints. To provide a self-contained view in this paper, Section 3.1 repeats the core definitions from [14], which form the basis for our approach. In Section 3.2, we introduce the textual RBAC DSL which allows to define entailment constraints in a simple textual syntax and enables a seamless mapping of UML-based process-related RBAC models (see [14]) to the implementation level. The core part of the textual RBAC DSL is based on [21]. For this paper, it has been extended with capabilities for the specification of entailment constraints.

3.1. Business Activity RBAC Models

Definition 1 (Business Activity RBAC Model). A Business Activity RBAC Model $BRM = (E, Q, D)$ where $E = S \cup R \cup P_T \cup P_I \cup T_T \cup T_I$ refers to pairwise disjoint sets of the metamodel, $Q = rh \cup tra \cup rsa \cup ptd \cup pi \cup ti \cup es \cup er$ to mappings that establish relationships, and $D = sb \cup rb \cup sme \cup dme$ to binding and mutual exclusion constraints, such that:

- For the sets of the metamodel:
  - An element of S is called *Subject*. $S \neq \emptyset$.
  - An element of R is called *Role*. $R \neq \emptyset$.
  - An element of P_T is called *Process Type*. $P_T \neq \emptyset$.
  - An element of P_I is called *Process Instance*.
  - An element of T_T is called *Task Type*. $T_T \neq \emptyset$.
  - An element of T_I is called *Task Instance*.

In the list below, we iteratively define the partial mappings of the Business Activity RBAC Model and provide corresponding formalizations ($\mathcal{P}$ refers to the power set, for further details see [14]):

1. The mapping $rh : R \mapsto \mathcal{P}(R)$ is called **role hierarchy**. For $rh(r_s) = R_j$ we call r_s *senior role* and R_j the set of direct *junior roles*. The transitive closure rh^* defines the inheritance in the role hierarchy such that $rh^*(r_s) = R_{j^*}$ includes all direct and transitive junior roles that the senior role r_s inherits from. The role hierarchy is cycle-free, i.e. for each $r \in R : rh^*(r) \cap \{r\} = \emptyset$.
2. The mapping $tra : R \mapsto \mathcal{P}(T_T)$ is called **task-to-role assignment**. For $tra(r) = T_r$ we call $r \in R$ *role* and $T_r \subseteq T_T$ is called the set of *tasks assigned to r*. The mapping $tra^{-1} : T_T \mapsto \mathcal{P}(R)$ returns the set of roles a task is assigned to (the set of roles owning a task).
   This assignment implies a mapping **task ownership** $town : R \mapsto \mathcal{P}(T_T)$, such that for each role $r \in R$ the tasks inherited from its junior roles are included, i.e. $town(r) = \bigcup_{r_{inh} \in rh^*(r)} tra(r_{inh}) \cup tra(r)$. The mapping $town^{-1} : T_T \mapsto \mathcal{P}(R)$ returns the set of roles a task is assigned to (directly or transitively via a role hierarchy).
3. The mapping $rsa : S \mapsto \mathcal{P}(R)$ is called **role-to-subject assignment**. For $rsa(s) = R_s$ we call $s \in S$ *subject* and $R_s \subseteq R$ the set of *roles assigned to this subject* (the set of roles owned by s). The mapping $rsa^{-1} : R \mapsto \mathcal{P}(S)$ returns all subjects assigned to a role (the set of subjects owning a role).
   This assignment implies a mapping **role ownership** $rown : S \mapsto \mathcal{P}(R)$, such that for each subject $s \in S$ all direct and inherited roles are included, i.e. $rown(s) = \bigcup_{r \in rsa(s)} rh^*(r) \cup rsa(s)$. The mapping $rown^{-1} : R \mapsto \mathcal{P}(S)$ returns all subjects assigned to a role (directy or transitively via a role hierarchy).
4. The mapping $ptd : P_T \mapsto \mathcal{P}(T_T)$ is called **process type definition**. For $ptd(p_T) = T_{p_T}$ we call $p_T \in P_T$ process type and $T_{p_T} \subseteq T_T$ the set of task types associated with p_T.
5. The mapping $pi : P_T \mapsto \mathcal{P}(P_I)$ is called **process instantiation**. For $pi(p_T) = P_i$ we call $p_T \in P_T$ *process type* and $P_i \subseteq P_I$ the set of process instances instantiated from process type p_T.
6. The mapping $ti : (T_T \times P_I) \mapsto \mathcal{P}(T_I)$ is called **task instantiation**. For $ti(t_T, p_I) = T_i$ we call $T_i \subseteq T_I$ set of *task instances*, $t_T \in T_T$ is called *task type* and $p_I \in P_I$ is called *process instance*.

7. Because role-to-subject assignment is a many-to-many relation (see Def. 1.3), more than one subject may be able to execute instances of a certain task type. The mapping $es : T_I \mapsto S$ is called **executing-subject** mapping. For $es(t) = s$ we call $s \in S$ the *executing subject* and $t \in T_I$ is called *executed task instance*.
8. Via the role hierarchy, different roles may posses the permission to perform a certain task type (see Def. 1.1 and Def. 1.2). The mapping $er : T_I \mapsto R$ is called **executing-role** mapping. For $er(t) = r$ we call $r \in R$ the *executing role* and $t \in T_I$ is called *executed task instance*.
9. The mapping $sb : T_T \mapsto \mathcal{P}(T_T)$ is called **subject-binding**. For $sb(t_1) = T_{sb}$ we call $t_1 \in T_T$ the *subject binding task* and $T_{sb} \subseteq T_T$ the set of *subject bound tasks*.
10. The mapping $rb : T_T \mapsto \mathcal{P}(T_T)$ is called **role-binding**. For $rb(t_1) = T_{rb}$ we call $t_1 \in T_T$ the *role binding task* and $T_{rb} \subseteq T_T$ the set of *role bound tasks*.
11. The mapping $sme : T_T \mapsto \mathcal{P}(T_T)$ is called **static mutual exclusion**. For $sme(t_1) = T_{sme}$ with $T_{sme} \subseteq T_T$ we call each pair $t_1 \in T_T$ and $t_x \in T_{sme}$ *statically mutual exclusive tasks*.
12. The mapping $dme : T_T \mapsto \mathcal{P}(T_T)$ is called **dynamic mutual exclusion**. For $dme(t_1) = T_{dme}$ with $T_{dme} \subseteq T_T$ we call each pair $t_1 \in T_T$ and $t_x \in T_{dme}$ *dynamically mutual exclusive tasks*.

3.2. RBAC Modeling for Business Processes

Figure 3 depicts the core RBAC metamodel and its connection with the core elements of the BusinessActivity metamodel. In particular, Figure 3 outlines how we extended our DSL from [21] to include process-related RBAC entailment constraints (see [14]). The different model elements are described below.

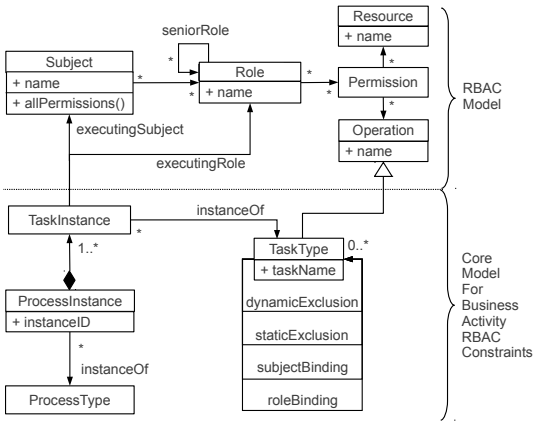


Figure 3: Excerpt of RBAC Metamodel and Business Activity Metamodel

A `ProcessInstance` has a unique `instanceID`, a `ProcessType`, and is composed of multiple `TaskInstance` objects which are again instances of a certain `TaskType`. The class `TaskType` has a `name` and four reflexive associations that define mutual exclusion and binding constraints. `Subjects` are identified by a `name` attribute and are associated with an arbitrary number of `Roles`, which are themselves associated with `Permissions` to execute certain `Operations`. A `TaskType` in the BusinessActivity metamodel corresponds to an `Operation` in the RBAC metamodel. Roles may inherit permissions from other roles (association `seniorRole`). In our approach, we directly associate Web service instances with `Resources`. That is, a subject that attempts to invoke a Web service operation *op* on a service resource *res* must be associated with a role that holds a permission to execute *op* on *res*. A detailed description of the BusinessActivity metamodel and corresponding OCL (Object Constraint Language) constraints can be found in [14]. We utilize the core parts of this model and focus on the mapping of the RBAC constraints to a textual DSL and to business process execution platforms, as illustrated for WS-BPEL in Section 5.

3.3. RBAC DSL Statements

Our RBAC DSL is implemented as an embedded DSL [27] and is based on the scripting language *Ruby* as host programming language. We now briefly discuss how the model elements are mapped to language constructs provided by the DSL (see also Section 3.1 and Figure 3). Table 1 lists the basic DSL statements (left column) and the corresponding effect (right column). In the table, keywords of the DSL syntax are printed in **`bold typewriter`** font, and placeholders for custom (scenario-specific) expressions are printed in *italics*.

RBAC DSL Statement	Effect
`RESOURCE` *name* [*description*]	Define new resource
`OPERATION` *name* [*description*]	Define new operation
`SUBJECT` *name* [*description*]	Define new subject
`ROLE` *name* [*description*]	Define new role
`ASSIGN` *subject role*	Assign role to subject
`INHERIT` *juniorRole seniorRole*	Let senior role inherit a junior role
`PERMIT` *role operation resource*	Allow a role to execute a certain operation on a specific resource
`TASK` *name operation resource*	Define operation-to-task mapping
`DME` *task1 task2*	Define dynamic mutual exclusion (DME)
`SME` *task1 task2*	Define static mutual exclusion (SME)
`RBIND` *task1 task2*	Define role-binding (RBind)
`SBIND` *task1 task2*	Define subject-binding (SBind)

Table 1: Semantics of RBAC DSL Statements

The RBAC DSL statements `RESOURCE`, `OPERATION`, `SUBJECT` and `ROLE` are used to create resources, operations, subjects and roles with the respective *name* and optional *description* attributes. `ASSIGN` creates an association between a subject and a role. `INHERIT` takes two parameters, a junior-role and a senior-role name, and causes the senior-role to inherit all permissions of the junior-role. `PERMIT` expresses the permission for a role to execute a certain operation on a resource. `DME` and `SME` allow the specification of dynamically or statically mutual exclusive operations. Using `RBIND` and `SBIND`, two operations are subjected to role-binding or subject-binding constraints. Finally, the `TASK` statement is used to establish

a mapping from our RBAC DSL to implementation level artifacts. More precisely, operations are mapped to concrete WS-BPEL tasks (see Section 5.2). The complete access control configuration for the patient examination scenario, expressed via RBAC DSL statements, is printed in Appendix A.

4. Model Transformations of Process Definitions for Runtime Constraint Enforcement

To enforce the RBAC constraints at runtime, the business process needs to follow a special procedure. If the process executes a secured task, it needs to provide a valid authentication token for the active user. For instance, this token contains information which subject (e.g., "Jane") executes an operation, and under which role (e.g. "Staff") this individual operates. In this section, we discuss our approach for automatically obtaining these authentication tokens to enforce security at runtime.

Figure 4 illustrates which artifacts are utilized by the instances of the business process. We follow the concepts of the SAML framework [34] and provide the authentication data with the aid of an Identity Provider (*IdP*) service. An IdP is a service provider that maintains identity information for users and provides user authentication to other services. The IdP is a reusable standard component; its sole responsibility is to authenticate the user and to issue an *AuthData* document which asserts the user's identity (subject and role). As such, the IdP has no knowledge about the process structure and RBAC constraints. Hence, we utilize the decoupled RBAC Manager Service which keeps track of the state of the process instances. The RBAC Manager Service knows the process structure and decides, based on the RBAC constraints, which subject or role is responsible for the next task (see also [19]).

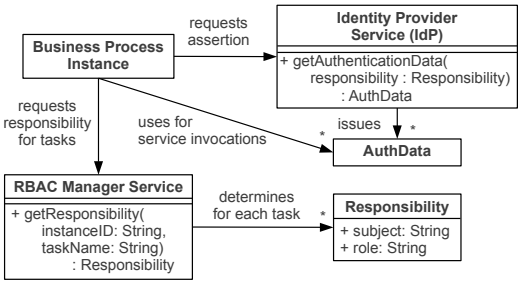


Figure 4: Relationship Between Business Process Instance and Security Enforcement Artifacts

Combining the functionality of `getResponsibility` and `getAuthenticationData` (see Figure 4) constitutes the core protocol for obtaining authentication tokens that enable the enforcement of task-based entailment constraints in a BusinessActivity. This recurring protocol is executed for each secured task; hence, it need not be implemented manually, but should ideally be generated automatically on top of the business process model that is defined by the developer. We therefore aim at providing automatic transformations to convert the domain-specific extensions for mutual exclusion and binding constraints in BusinessActivity models into regular activity models which perform the required IAM tasks. This transformation is required as an intermediate step towards the generation of corresponding definitions that are directly deployable and executable (e.g., by WS-BPEL engines). In the following, we describe the transformation procedure in detail and discuss different implementation and runtime aspects.

4.1. Model Transformations to Enforce Mutual Exclusion Constraints

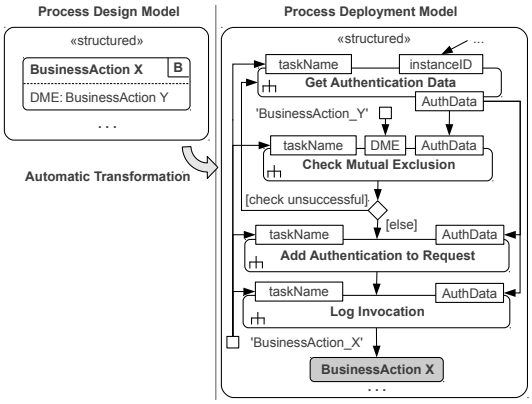
Here we discuss the detailed procedure for runtime enforcement of mutual exclusion constraints in the form of DME and SME tasks. We propose an approach for transforming design-time BusinessActivity models into deployable standard activity models that comply with this procedure. The transformations for enforcing mutual exclusion constraints are illustrated in Figure 5. Tasks representing invocations to external Web services are printed in grey, while structured activities and tasks with local processing logic are depicted with a white background.

The transformed activity models with mutual exclusion constraints in Figure 5 contain four additional tasks. All four tasks are UML *CallBehaviorAction*s [32] (indicated by the rake-style symbol) which consist of multiple sub-tasks. The internal processing logic depends on the concrete target platform; later in Section 5.1 we discuss the detailed logic for WS-BPEL.

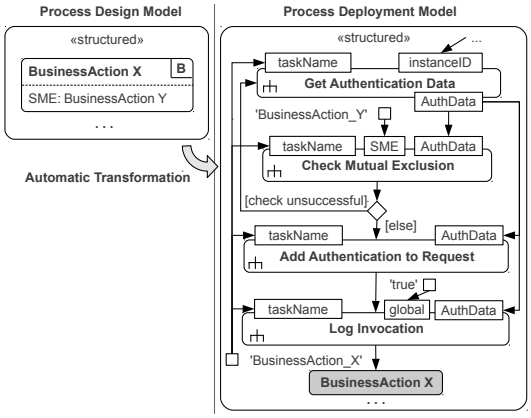
The task *Get Authentication Data* invokes the IdP service to obtain the authentication data token (*AuthData*) to be used for later invocation of the BusinessAction. The second inserted task is *Check Mutual Exclusion*, which is responsible for checking whether the provided authentication data are valid with respect to the mutual exclusion constraint. A UML value pin [32] holding the name of the corresponding task provides the input for the pin *DME* (Figure 5(a)) or the pin *SME* (Figure 5(b)), respectively. Additionally, the *Check Mutual Exclusion* task receives as input the name of the task to-be-executed (*taskName*, which is known from the original process definition), and the *AuthData* (received from the IdP service). The decision node is used to determine whether *Check Mutual Exclusion* has returned a successful result. If the result is unsuccessful (i.e., a constraint violation has been detected) the control flow points back to *Get Authentication Data* to ask the IdP again for a valid authentication data token. Otherwise, if the result is successful, the task *Add Authentication to Request* appends the user credentials in *AuthData* to the request message for the target Web service operation. The fourth inserted task is *Log Invocation*, which adds a new log record that holds the name of the task (*taskName*) and the *AuthData* of the authenticated user. The input pin *global* determines whether the log entry is stored in a local variable of the process instance (value *null*) or in a global variable accessible from all process instances (value *'true'*).

4.2. Model Transformations to Enforce Binding Constraints

The approach for transforming binding constraints in BusinessActions (illustrated in Figure 6) is similar to the transformation for mutual exclusion constraints presented in Sec-

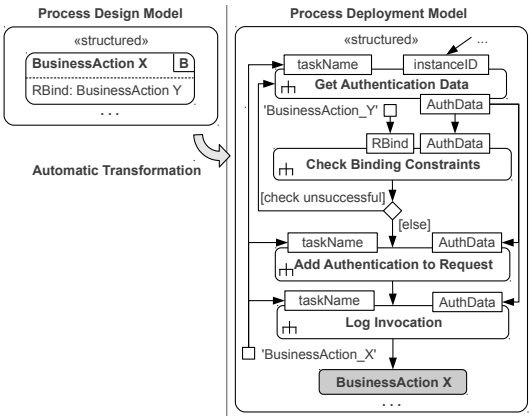


(a) Transformation for DME Constraints

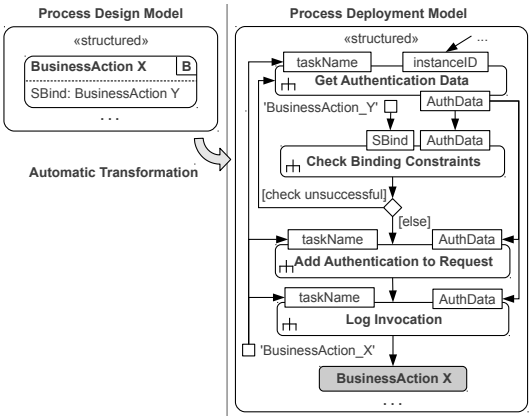


(b) Transformation for SME Constraints

Figure 5: Process Transformations to Enforce Mutual Exclusion Constraints



(a) Transformation for Role Binding



(b) Transformation for Subject Binding

Figure 6: Process Transformations to Enforce Binding Constraints

tion 4.1. The transformed process model first requests authentication data from the IdP service. The task *Check Binding Constraints* then checks the resulting *AuthData* with respect to role-bindings (*RBind*, Figure 6(a)) and subject-bindings (*SBind*, Figure 6(b)). The process asks for new user credentials and repeats the procedure if the binding constraint is not fulfilled.

Note that the entailment constraints are checked directly inside the process, not by the IdP service. Even though the AuthData (subject, role) obtained from the IdP is trusted and assumed to properly represent the user executing the process, the AuthData may be invalid with respect to entailment constraints. Hence, the branch "check unsuccesful" indicates that the process instance asks for a different user to login and execute the task. As the log of previous tasks is stored locally by each process instance (except for SME constraints, where log entries are also stored globally), the *Check Binding* and *Check Mutual Exclusion* tasks are required directly inside the process logic and are not outsourced to external services. This approach is able to deal with deadlock situations (evaluated in Section 7.2).

In certain deployments, the platform providers (e.g., hospital management) may be interested in tracking failed authorizations. For brevity, such mechanisms are not included in Figures 5 and 6, but extending the approach with notifications is straight-forward.

4.3. Transformation Rules for Combining Multiple Constraints

So far, the transformation rules for the four different types of entailment constraints in BusinessActivities (role-binding, subject-binding, SME, DME) have been discussed in isolation. However, as the scenario in Section 2 illustrates, BusinessActions can possibly be associated with multiple constraints (e.g., *Get Critical History*). Therefore, we need to analyze how the transformation rules can be combined while still maintaining the constraints' semantics. A simple approach would be to successively apply the atomic transformations for each BusinessAction and each of the constraints associated with it. However, this approach is not suited and may lead to incorrect re-

sults. For instance, if we consider the task *Get Critical History* with the associated DME and SBind constraints, the process might end up requesting the authentication data twice, which is not desired. Therefore, multiple constraints belonging to the same task are always considered as a single unit (see also [19]).

Figure 7 depicts the transformation template for a generic sample BusinessAction X with multiple constraints $c_1, c_2, \ldots, c_n$.

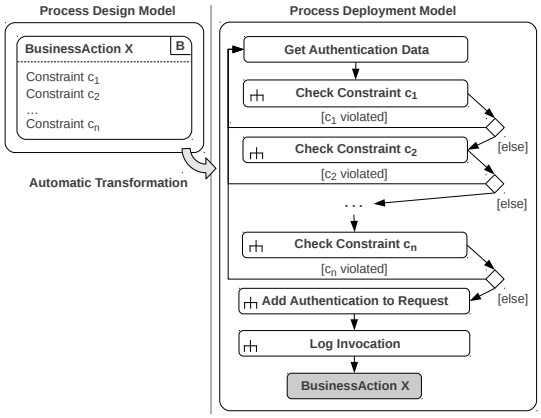


Figure 7: Generic Transformation Template for Business Action With Multiple Constraints

5. Application to SOA and WS-BPEL

This section discusses details of the process transformation from Section 4 and illustrates how the approach is applied to SOA, particularly WS-BPEL and the Web services framework.

5.1. Supporting Tasks for IAM Enforcement in WS-BPEL

In the following we discuss the internal logic of the five supporting IAM tasks used in the transformed activity models for the enforcement of mutual exclusion (Section 4.1) and binding constraints (Section 4.2).

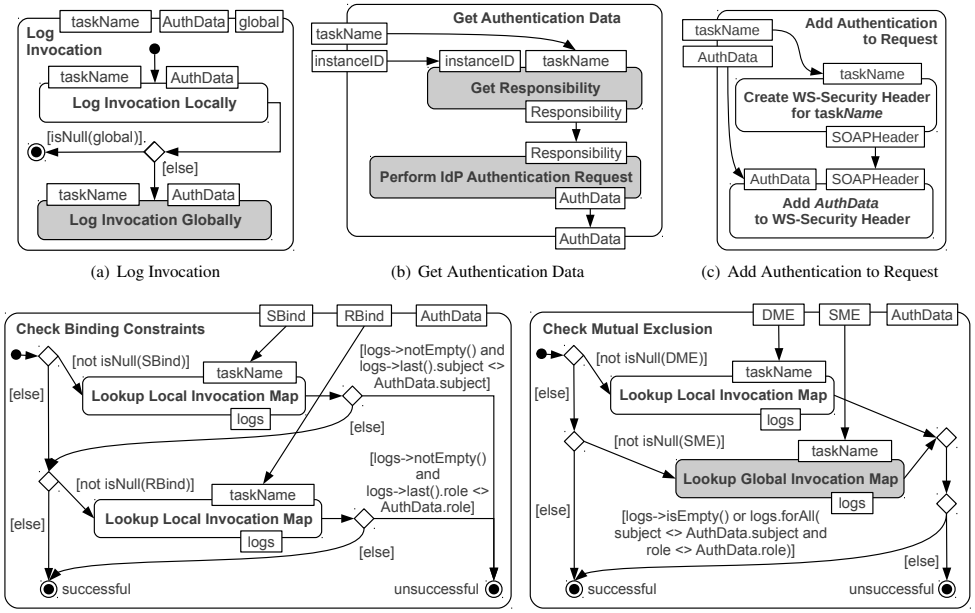
Task **Log Invocation**: In general, process-related RBAC constraints rely on knowledge about historical task executions (see also [14]). Therefore, a mechanism is required to store data about previous service invocations. One conceivable approach is that the process execution engine keeps track of the invocation history. To that end, invocation data can be stored either in a local variable of the process instance (for DME constraints) or in a global variable that is accessible from all process instances (for SME constraints). Unfortunately, WS-BPEL does not support global variables, but we can overcome this issue by using an external logging Web service. Figure 8(a) shows the *Log Invocation* activity, which stores data about service calls, including the name of the invocation and the *AuthData* of the user under which the process executes. The invocation is first stored in a local array variable of WS-BPEL. If the input pin named *global* is not null, the data is also stored with the external logging service (*Log Invocation Globally*). Currently, our framework relies on a central logging service. As part of our future work, we tackle advanced challenges such as privacy, and timing issues that come with decentralized logging.

Task **Get Authentication Data**: This supporting IAM task is used to obtain authentication tokens, see Figure 8(b). The identifier of the affected process task is provided as a parameter *taskName*. For instance, in the case of WS-BPEL, the `name` attribute of the corresponding `invoke` statement can be used to determine this value. As outlined in Section 4, the procedure is split up between the RBAC Manager service and the IdP. First, the invocation *Get Responsibility* asks the RBAC Manager for the role or subject responsible for executing the next task. All combinations of values are possible, i.e., either subject or role, or both, or none of the two may be specified. The subject/role responsibility information is used to execute an *IdP Authentication Request*. The authentication method performed by the IdP is transparent; for instance, it may perform smartcard based authentication or ask for username and password. The *AuthData* output pin provided by this invocation contains the definite subject and role name of the user.

Task **Add Authentication to Request**: The activity in Figure 8(c) illustrates how authentication data are appended to the invocation of Business Actions. First, the *AuthData* information is used to request a SAML assertion from the IdP service. This token contains the subject and role with a trusted signature that ensures the integrity of the assertion content. The assertion is then added to the request message for the target service operation (the name is specified via the input pin *taskName*) using the SOAP header mechanism [35] (SOAP is the communication protocol used by Web services). Note that this activity leaves room for optimization. If many tasks in the process are executed by the same subject and role, it is advantageous to cache and reuse the SAML tokens in a local variable of the process instance. However, caching security tokens carries the risk of inconsistencies if the RBAC policies change.

Task **Check Binding Constraints**: Figure 8(d) contains the activity *Check Binding Constraints*, whose internal logic is to check the logged invocations with role-binding and subject-binding against the *AuthData* information. If the *SBind* parameter is set, the activity looks up the last corresponding log entry (the *taskName* of the log entry needs to be equal to *SBind*) in the local invocation map of the WS-BPEL process instance. If the returned array (named *logs*) is not empty, then the subject stored in the last log entry needs to be identical to the subject in *AuthData*. Analogously, if the *RBind* parameter is set, then the role of the last log entry with *taskName* equal to *RBind* must be equal to the role in *AuthData*. If and only if all conditions hold true, the activity returns a success status.

Task **Check Mutual Exclusion**: Similarly, the *Check Mutual Exclusion* activity in Figure 8(e) uses the log data to check the *AuthData* against the previously performed invocations. If the input parameter *DME* is set, WS-BPEL looks up the log entries from the local invocation map. Otherwise, if an *SME* parameter is provided, the corresponding logs are received from the external logging service (global invocation map). The activity



(a) Log Invocation (b) Get Authentication Data (c) Add Authentication to Request

(d) Check Binding Constraints (e) Check Mutual Exclusion

Figure 8: Supporting Tasks for IAM Enforcement in WS-BPEL

returns a successful result if either the *logs* sequence is empty or all log entries have a different subject and role than the given *AuthData*. Due to the possibly large number of entries in the *logs* sequence, it is crucial where these conditions are evaluated (by the process or the logging service directly). To avoid transmitting log data over the network, we recommend the implementation variant in which the logging service itself validates the conditions. To that end, *AuthData* is sent along with the request to the logging service and the service returns a boolean result indicating whether the constraints are satisfied.

5.2. RBAC DSL Integration with WS-BPEL

The `TASK` statement of the RBAC DSL realizes a mapping from operations to concrete WS-BPEL tasks (`invoke` activities). This corresponds to the model in Figure 3, where `TaskType` in the Business Activities metamodel is mapped to `Operation` in the RBAC metamodel. Using this mapping, we are able to automatically apply all Business Activity entailment constraints to the corresponding WS-BPEL `invoke` activities.

DSL Statement	WS-BPEL DSL Statement
DME *task*1 *task*2	`<invoke name="`*task*1`" rbac:dme="`*task*2`" ../>`
SME *task*1 *task*2	`<invoke name="`*task*1`" rbac:sme="`*task*2`" ../>`
SBIND *task*1 *task*2	`<invoke name="`*task*1`" rbac:sbind="`*task*2`" ../>`
RBIND *task*1 *task*2	`<invoke name="`*task*1`" rbac:rbind="`*task*2`" ../>`

Table 2: Mapping of RBAC DSL Statements to WS-BPEL DSL Statements

In our approach, WS-BPEL `invoke` activities are constrained using specialized DSL statements. The DSL uses the extension mechanism of WS-BPEL and introduces new XML attributes `rbac:dme`, `rbac:sme`, `rbac:sbind` and `rbac:rbind` (the prefix `rbac` refers to the XML namespace these attributes are part of). These attributes are then directly annotated to the `invoke` activities in WS-BPEL. Table 2 illustrates how the relevant RBAC DSL statements are mapped to the corresponding WS-BPEL DSL statements. For instance, the `DME` statement is mapped to a `rbac:dme` attribute. The parameters of the DSL statements in Table 2 refer to the task types defined using the `TASK` statement (see Section 3.3). Note that these `rbac:`$\star$ attributes can be multi-valued. That is, multiple values can be separated by commas. For example, a task that is dynamically mutual exclusive to *task*1 and *task*2 can be annotated with a `rbac:dme="task1,task2"` attribute.

5.3. Automatic Transformation of WS-BPEL Definition

At deployment time, the business process model is automatically transformed to ensure correct enforcement of identity and access control policies at runtime. The transformation can happen on different abstraction levels, either based on the platform-independent model (PIM) or on the platform-specific model (PSM) (see, e.g., [36]). On the PIM level, model transformation languages such as *Query/View/Transformation* (QVT) [37] can be used to perform UML-to-UML transformation of process activity models. Our approach proposes a transformation directly on the PSM model, i.e., the WS-BPEL process definition file.

Algorithm 1 WS-BPEL Transformation Algorithm

Input: WS-BPEL document *bpel*, Fragment Templates *tmpl*
Output: transformed WS-BPEL document

1: add `<import ../>`, `<partnerLink ../>`, and `<variable ../>` statements to *bpel*
2: add `<assign ../>` statements to initialize *ProcessInstanceID* and *InvocationLogs* variables
3: **for all** *bpel*`//invoke` as *inv* **do**
4: **if** *inv*`/@rbac:*` **then**
5: *authInvoke* ← create `<invoke ../>` for operation *getAuthenticationData* and partnerLink *IdP*
6: *constraintChecks* ← ∅
7: **for all** *inv*`/@rbac:*` as *constraint* **do**
8: *tasks* ← split value of *constraint* by commas
9: **for all** *tasks* as *task* **do**
10: *check* ← create `<if>..</if>` which checks outcome of *authInvoke* for RBAC entailment constraint *constraint* and task *task*
11: *constraintChecks* ← *constraintChecks* ∪ *check*
12: **end for**
13: **end for**
14: *enforcementBlock* ← wrap sequence *authInvoke*‖*constraintChecks* in new `<while>..</while>` block
15: insert *enforcementBlock* before *inv*
16: **if** *inv*`/@rbac:sme` **then**
17: *logInvoke* ← create `<invoke ../>` for operation *logInvocation* via partnerLink *LoggingService*
18: insert *logInvoke* after *inv*
19: **end if**
20: **end if**
21: **end for**

Algorithm 1 gives a simplified overview of which WS-BPEL code fragments are injected, and where. Variable names are printed in *italics*, and XML markup and XPath expressions are in `typewriter` font. The input is a WS-BPEL document *bpel* with security annotations. Firstly, various required documents (e.g. the XSD files of SAML and WS-Security) need to be imported into the WS-BPEL process using `import` statements. Then the `partnerLink` declarations for the needed services (such as the IdP service) are added to *bpel*, and `variable` declarations are created (e.g. input/output variables for `getAuthenticationData` operations). Using `assign` statements, some variables (such as `ProcessInstanceID`) are initialized. Next, the algorithm loops over all `invoke` elements that have an attribute from the `rbac` namespace assigned (e.g. `rbac:rbind` or `rbac:dme`). For every matching `invoke` several WS-BPEL code injections and transformations have to be conducted. Firstly, an `invoke` statement (`authInvoke`) is created. At runtime, this statement calls the `IdP`'s `getAuthenticationData` operation. Next, an empty set (`constraintChecks`) is created. Afterwards, the algorithm iterates over all constraints (e.g. `rbac:sbind`) that have been defined for this particular `invoke` statement. The values of every `constraint` are split by commas. For instance, in the case of an `rbac:dme="task1,task2"` annotation, `constraint` is `rbac:dme` and `tasks` is a set with two elements (`task1` and `task2`). For every `task` an `if`-block (`check`) is created. At runtime, this `if`-block checks, if there is a violation of the entailment constraint `constraint` regarding another task `task`. Every `check` added to the set `constraintChecks`. Next, a new `<while>..</while>`-block (`enforcementBlock`) is created. This block envelopes the previously created `authInvoke` statement and all checks contained in `constraintChecks`. Finally, this `enforcementBlock` is inserted directly before the secured `invoke` statement. Just in case the latter is also annotated using a `rbac:sme` attribute, an additional invocation is injected right after the actual `invoke` element. This one calls the `logInvocation` operation via the `LoggingService` PartnerLink.

6. Implementation

In this section, we discuss our prototype implementation of the proposed approach. The implementation is integrated in the SeCoS[1] (*Se*cure *Co*llaboration in *S*ervice-based systems) framework. This section is divided into four parts: firstly, we outline the architecture of the system and the relationship between the individual services and components in Section 6.1; secondly, the SAML-based SSO mechanism is described in Section 6.2; in Section 6.3 we present the algorithm for automatic transformation of WS-BPEL definitions containing security annotations from our DSL; finally, Section 6.4 discusses the implementation for checking constraints over the log data.

6.1. System Architecture

Figure 9 sketches the high-level architecture and relationships between the example process and the system components.

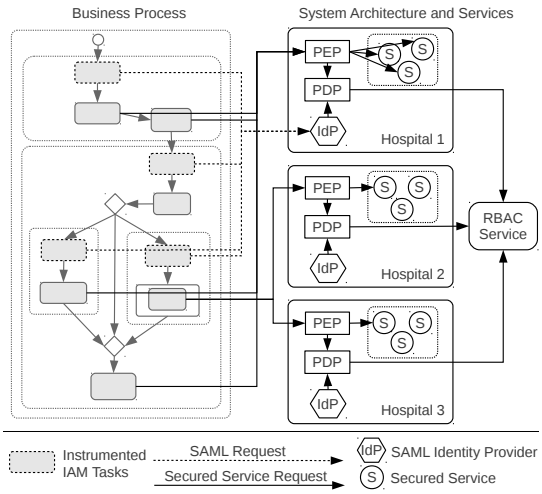


Figure 9: Example Process in System Architecture

The patient examination scenario from Section 2 is implemented using WS-BPEL and deployed in a Glassfish[2] server.

[1] http://www.infosys.tuwien.ac.at/prototype/SeCoS/
[2] https://glassfish.dev.java.net/

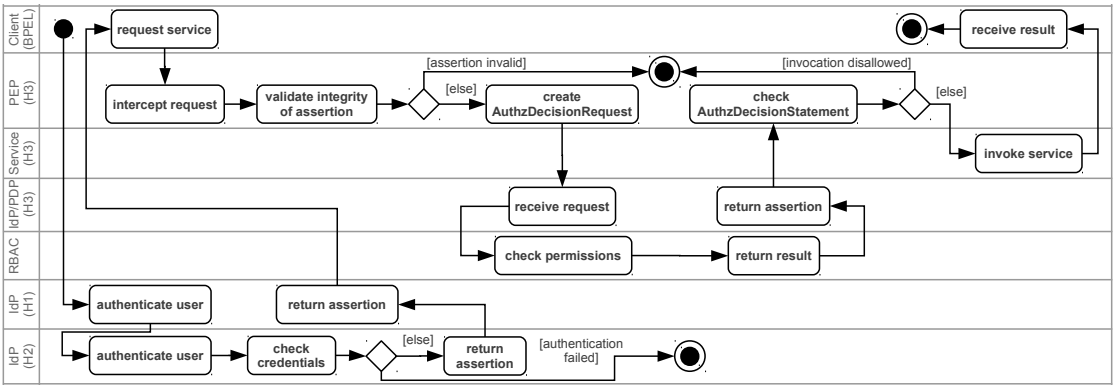


Figure 10: Identity and Access Control Enforcement Procedure

The scenario involves three hospitals, which host the protected services for patient management and examination. All service invocations are routed through a Policy Enforcement Point (**PEP**), which acts as a central security gateway, intercepts every incoming service request and either allows or disallows its invocation. It is important that the PEP operates transparently and as close to the protected resources (i.e., services) as possible. Using the Java API for XML Web services (JAX-WS), the PEP has been implemented as a SOAP message handler (interface `SOAPHandler`). This handler can be plugged into the Web service's runtime engine in a straightforward manner. Once activated, the interceptor is able to inspect and modify inbound and outbound SOAP messages and to deny service invocations.

Each hospital runs a SAML **IdP** service, which is used to issue the SAML assertions that are required in the WS-BPEL process. The IdP's responsibility is twofold: firstly, it authenticates users; secondly, the IdP assures the identity of a subject and its associated attributes (e.g., roles) by issuing a SAML assertion SOAP header which is used in subsequent service invocations. For the sake of an easy integration into the given system environment, we decided to use the JAX-WS API for implementing the Login Web service. This SOAP Web service offers a `login` method. It requires a username/password pair and returns a SAML assertion. Internally, we utilize the Java Architecture for XML Binding (JAXB) for parsing and creating SAML assertions. Additionally, the Apache XML Security for Java[3] library is used for digitally signing XML documents (i.e., the SAML assertions).

The actual decision whether an invocation should be prevented or not is typically delegated to another entity, the Policy Decision Point (**PDP**). When deciding over the access to a service resource the PDP has to make sure that the subject attempting to access the resource has the permission to do so. This decision is based on the policy information stored in the RBAC repository (which is based on the DSL statements authored by domain experts). In our implementation, the core functionality of the PDP is embedded into the RBAC DSL (see Section 3.2). That is, the DSL offers an `access` method that can be used to determine whether the requesting subject is permitted to access the target resource (service) under the specified context and role (see Figure 9). In order to make this functionality accessible to the outside of the DSL's interpreter, we developed a RESTful Web service, that bridges HTTP requests to the interpreter. More precisely, the PDP service uses the Bean Scripting Framework (BSF)[4] to access the interpreter. The Java API for RESTful Web Services (JAX-RS) is used to realize the PDP service's RESTful Web interface.

6.2. SAML-based Single Sign-On

Figure 10 depicts an example of the Identity and Access Control enforcement procedure modeled in UML. To illustrate the SSO aspect of the scenario, we assume that a patient with subject name "Alice" (cf. Figure 3), who is registered in hospital 2 (H2), is examined in hospital 1 (H1) and requests her patient history from previous examinations in hospital 3 (H3). The procedure is initiated by the WS-BPEL process which requests the execution of a protected Web service.

Prior to issuing the actual service request, the user has to authenticate using the SAML IdP. The IdP queries the user database to validate the credentials provided by the client. As the credentials of user *Alice* are not stored in the DB of H1, the IdP contacts the IdP of H2, which validates the credentials.

If the user credentials could not be validated, the process is terminated prematurely and a SOAP fault message is returned. In our example scenario, the business process receives the fault message and activates corresponding WS-BPEL fault handlers. Otherwise, if the credentials are valid, the IdP creates a signed assertion similar to the one shown in Listing 1 and passes it back to the WS-BPEL process (see Figure 10). The business process attaches the assertion to the actual service request.

[3] http://santuario.apache.org/

[4] http://commons.apache.org/bsf/

```
<Assertion>
  <Issuer>http://h2.com/IdP</Issuer>
  <ds:Signature>...</ds:Signature>
  <Subject><NameID>Alice</NameID></Subject>
  <Conditions NotBefore="2012-05-17T09:48:36.171Z"
              NotOnOrAfter="2012-05-17T10:00:36.171Z"/>
  <AttributeStatement>
    <Attribute Name="role">
      <AttributeValue>staff</AttributeValue>
    </Attribute>
  </AttributeStatement>
</Assertion>
```

Listing 1: Exemplary SAML Assertion Carrying Subject and Role Information

The example SAML assertion in Listing 1 illustrates the information that is encapsulated in the header token when the scenario process invokes the `getPatientHistory` operation of the patient Web service of H3. The assertion states that the subject named `Alice`, which has been successfully authenticated by the IdP of the hospital denoted by the `Issuer` element (H2), is allowed to use the the role `staff` in the context `default`. The included XML signature element ensures the integrity of the assertion, i.e., that the assertion content indeed originates from the issuing IdP (H2) and has not been modified in any way. When the PEP of H3 intercepts the service invocation with the SAML SOAP header, its first task is to verify the integrity of the assertion. The signature verification requires the public key of the IdP that signed the assertion; this key is directly requested from the corresponding IdP (under `http://h2.com/IdP`) using SAML Metadata [38]. Our implementation uses the Apache XML Security for Java library to conduct the signature verification.

```
<Assertion>
  <Issuer>http://h3.com/IdP</Issuer>
  <ds:Signature>...</ds:Signature>
  <Subject>
    <NameID>Alice</NameID>
  </Subject>
  <AuthzDecisionStatement Decision="Permit"
    Resource="http://h3.com/patient">
    <Action>getPersonalData</Action>
  </AuthzDecisionStatement>
</Assertion>
```

Listing 2: Exemplary SAML Authorization Decision

After the PEP of H3 has verified the message integrity, it needs to determine whether the subject is authorized to access the requested service operation. This is achieved by the PDP service of H3 that allows the PEP to post a SAML Authorization Decision Query. The PDP answers this query by returning an assertion containing a SAML Authorization Decision Statement. Listing 2 shows an example SAML assertion which informs the PEP that our staff member is allowed to invoke the action (operation) `getPersonalData` of the resource (Web service) `http://h1.com/patient`. Analogously to the IdP service, we also used the JAX-WS API to implement the SOAP-based interface of the PDP service. The PDP offers the method `query`, which takes an Authorization Decision Query message as argument and returns an Authorization Decision Statement. Again, we leverage JAXB for parsing the SAML documents.

6.3. Automatic Transformation of WS-BPEL Definition

Since both WS-BPEL and SAML are XML based standards, we are able to reuse and utilize the broad line-up of existing XML tooling. The transformation procedure of WS-BPEL process definitions is hence based on XSLT (Extensible Stylesheet Language Transformations) [39], a language for arbitrary transformation and enrichment of XML documents.

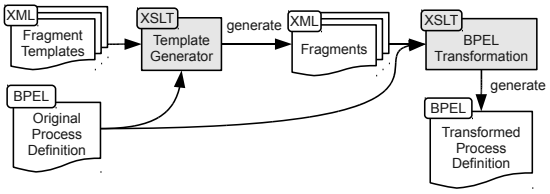


Figure 11: Artifacts of the Transformation Process

In general, the original WS-BPEL process is transformed by enriching the process definition file with code fragments that perform the IAM tasks (cf. Section 5.1). In principle, these fragments are generic and static, i.e., for arbitrary WS-BPEL processes nearly the same fragments can be injected. However, some fragments contain volatile elements that are specific to every single WS-BPEL process. As these fragments need to be adapted to fit a specific WS-BPEL process, we propose a two-stage transformation process. Figure 11 depicts an overview of the document artifacts involved in the transformation process, as well as the flow relations between them. The leftmost part of the figure indicates how the original WS-BPEL process definition file and various XML fragment files serve as input for the *Template Generator* XSLT file. This Template Generator constitutes the first transformation step and turns the generic fragment templates into fragments tailored to the target process definition. The last transformation step injects the generated fragments into the original WS-BPEL process file.

6.4. Checking Business Activity Constraints

The process transformation approach presented in Section 4 ensures runtime enforcement of Business Activity entailment constraints. For highly business- or security-critical systems we propose log analysis to additionally monitor that the process instances behave as expected (see, e.g., [40]). To check whether all constraints are fulfilled in the log data, we require an engine capable of querying the state of historical invocation data. As our framework is operating in a Web Services environment, XML is the prevalent data format and we focus mainly on XML tooling. We hence utilize XQuery [41] to continuously perform queries over the invocation logs stored in XML. To facilitate the handling of these queries, we use WS-Aggregation [42], a platform for event-based distributed aggregation of XML data.

Listing 4 prints exemplary log data that are emitted by the transformed business process and handled by WS-Aggregation. Each *log* element in the listing represents one invocation event.

```
1  let $cons := <constraints>
2                 <rbind> <task>Get_Personal_Data </task> <task>Assign_Physician </task> </rbind>
3                 <sbind> <task>Decide_On_Treatment </task> <task>Get_Critical_History </task> </sbind>
4                 <dme> <task>Get_Critical_History </task> <task>Get_Expert_Opinion </task> </dme>
5                 <sme> <task>Get_Expert_Opinion </task> <task>Get_Patient_History </task> </sme>
6                  ...
7               </constraints>
8  let $logs := //log
9
10 SME Tasks:
11 every $sme in $cons/sme, $t1 in $sme/task, $t2 in $sme/task satisfies ( $t1 = $t2 or
12    (every $i in $logs[@taskName=$t1] satisfies (
13       not(exists($logs[@taskName=$t2][@role=$i/@role]))
14          and
15       not(exists($logs[@taskName=$t2][@subject=$i/@subject])))))
16 DME Tasks:
17 every $dme in $cons/dme, $t1 in $dme/task, $t2 in $dme/task satisfies ( $t1 = $t2 or
18    (every $i in $logs[@taskName=$t1] satisfies (
19       not(exists($logs[@taskName=$t2][@subject=$i/@subject][@instanceID=$i/@instanceID])))))
20 Subject-Binding:
21 every $sbind in $cons/sbind, $t1 in $sbind/task, $t2 in $sbind/task satisfies ( $t1 = $t2 or
22    (every $i in $logs[@taskName=$t1] satisfies (
23       every $j in $logs[@taskName=$t2][@instanceID=$i/@instanceID] satisfies $i/@subject = $j/@subject)))
24 Role-Binding:
25 every $rbind in $cons/rbind, $t1 in $rbind/task, $t2 in $rbind/task satisfies ( $t1 = $t2 or
26    (every $i in $logs[@taskName=$t1] satisfies (
27       every $j in $logs[@taskName=$t2][@instanceID=$i/@instanceID] satisfies $i/@role = $j/@role)))
```

Listing 3: XQuery Assertion Expressions for Enforcing Business Activity Constraints

```
1  <log taskName="Get_Personal_Data" subject="john"
2       role="staff" instanceID="i1" time="1316423654600"/>
3  <log taskName="Assign_Physician" subject="john"
4       role="staff" instanceID="i1" time="..."/>
5  <log taskName="Get_Personal_Data" subject="john"
6       role="staff" instanceID="i2" time="..."/>
7  <log taskName="Get_Critical_History" subject="bob"
8       role="physician" instanceID="i1" time="..."/>
9  ...
```

Listing 4: Format of Invocation Data Logged as Events

Listing 3 prints the constraint enforcement queries, expressed as XQuery assertion statements that are expected to always yield a boolean *true* value. Lines 1-7 contain an excerpt of the constraint definitions in our scenario. For instance, the two tasks named *Get_Personal_Data* and *Assign_Physician* are in a role-binding relationship and hence combined in an element `rbind`. Moreover, the code binds the log elements from Listing 4 to the variable `$logs` (line 8). Finally, Listing 3 contains the four XQuery expressions used for enforcing constraints concerning SME tasks (lines 11-15), DME tasks (lines 17-19), subject-bindings (lines 22-25) and role-bindings (lines 27-30).

The four expressions use universal quantification (`every...in...satisfies`) to express assertions about pairs of tasks defined in the constraints list. The variables $*t1* and $*t2* refer to the names of the respective tasks. The query for SME loops over all pairs of SME tasks and ensures that the logs do not contain invocations for both tasks that use the same subject or the same role. The DME query tasks is similar, with the difference that only the subject is queried and additionally the *instanceID* attribute of the log entries is considered. Subject-binding is checked by ensuring that for all log entries of a particular process instance two tasks $*t1* and $*t2* are executed by the same subject. The role-binding query works analogously, but instead of using the *subject* attribute, here we require the *role* attribute to match for all *rbind*-connected tasks that occur in the same process instance.

7. Evaluation and Discussion

In this section, we evaluate various aspects to highlight the benefits, strengths, and weaknesses of the presented solution. Five business processes with entailment constraints were selected to conduct the evaluation, including our example process from Section 2 and four additional processes from existing literature. The examples represent typical processes from different domains and cover all constraint types supported by our approach. The key properties of the evaluated processes are summarized in Table 3: *ID* identifies the process (*P*1 is our sample process), $|T_T|$ is the total number of task types per process, $|CT_T|$ is the number of task types associated with entailment constraints[5], $|R|$ is the number of roles defined in the scenario, $|S|$ is the number of subjects used for the test, and $|HR|$ is the number of senior-junior relationships in the role hierarchy[6].

ID	Name	$\|T_T\|$	$\|CT_T\|$	$\|R\|$	$\|S\|$	$\|HR\|$
P1	Patient Examination	7	6	3	4	1
P2	Purchase Order [43]	6	4	2	3	1
P3	Paper Review [14]	5	4	3	5	0
P4	Tax Refund [16]	5	4	2	5	0
P5	Credit Application [14]	5	3	2	4	1

Table 3: Characteristics of Business Processes Used in the Evaluation

Although not all results of our evaluation are fully generalizable, they are arguably valid for a wide range of scenarios and SOA environments in general. An evident observation is that runtime enforcement of security constraints is computationally intensive, and therefore performance effects need to be taken into account. We also show that the proposed DSL greatly simplifies development of security-enabled WS-BPEL processes,

[5] $CT_T = \{\, t \in T_T \mid sb(t) \neq \emptyset \vee rb(t) \neq \emptyset \vee sme(t) \neq \emptyset \vee dme(t) \neq \emptyset \,\}$

[6] $HR = \{\, (s, j) \in R \times R \mid j \in rh(s) \,\}$

which becomes apparent when comparing the number of code artifacts before and after automatic transformation. However, the approach also has certain limitations which we also want to document explicitly. Overall, our evaluation is organized in four parts: first, we evaluate the runtime performance in Section 7.1; second, in Section 7.2 we verify the behavior of secured processes when provided with valid and invalid authentication data[7]; third, Section 7.3 evaluates the WS-BPEL transformation procedure; fourth, in Section 7.4 we discuss current limitations in the framework and general threats to validity. The experiments in Sections 7.1, 7.2 and 7.3 were executed on a machine with Quad Core 2.8GHz CPU, 8GB RAM, running Ubuntu Linux 9.10 (kernel 2.6.31-23).

7.1. Performance and Scalability

For our scalability evaluation we have defined, deployed, and executed different process instantiations (based on the example in Section 2) in a Glassfish server (version 2.1.1) with WS-BPEL engine (version 2.6.0). Here, we are only interested in the net processing time of the Web service invocations, the duration of human tasks is not considered. Therefore, the execution of business operations (e.g., *Obtain X-Ray Image* or *Decide On Treatment*) has zero processing time in our testbed.

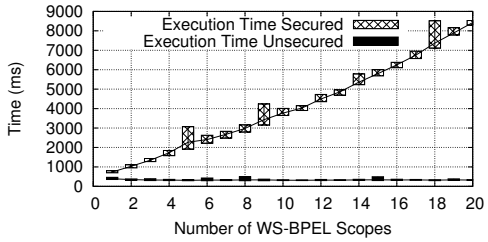


Figure 12: Process Execution Times – Secured vs Unsecured

The WS-BPEL process has been deployed in different sizes (multiple `scopes`, one `invoke` task per scope), once with enforced security (i.e., annotated with security attributes, automatically transformed at deployment time), and once in an unsecured version. The deployed processes were executed 100 times and we have computed the average value to reduce the influence of external effects. Figure 12 plots the execution time (minimum, maximum, average) for both the secured (top line) and the unsecured version (bottom line). The top/bottom of each box represents the maximum/minimum, respectively, and a trendline is drawn for the average value[8]. We observe that a single BusinessAction invocation in the unsecured version is very fast, whereas the secured version incurs a considerable overhead. The overhead is hardly surprising considering that for each business logic service the process needs to invoke the IdP and RBAC services, as well as apply and check XML signatures. However, the measured results indicate that the current implementation has potential room for additional optimization.

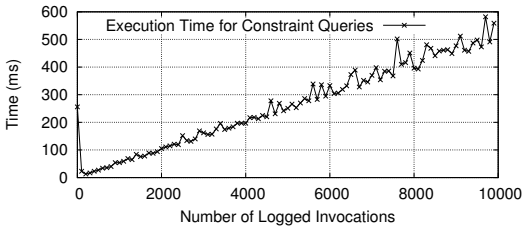


Figure 13: Execution Time of Constraint Queries for Increasing Log Data

In addition to the end-to-end performance of the secured WS-BPEL process, we also evaluated the performance of enforcing the BusinessActivity constraints using the XQuery based querying approach. To that end, we stored 10000 entries with SME, DME, SBind and RBind constraints to the invocation log and measured the time required to execute the four constraint queries in Listing 3. The results are illustrated in Figure 13, which plots the time for every 100th invocation over time. As the testbed started cleanly from scratch, the first logged invocation(s) took longer (~250ms) because of internal initialization tasks in the log store and the WS-Aggregation query engine. Starting from the second data point (invocation 100), we see the query time increasing by around 6ms per 100 queries. To provide an insight about resource consumption, the CPU utilization and Java heap space usage are plotted in Figure 14. The slight fluctuations in heap space are due to Java's garbage collection procedure. The four constraint queries are executed in parallel, but since they access a shared data structure with log data, internal thread synchronization is applied. Hence, CPU utilization reaches only a peak value of ~70% (i.e., 3 of the 4 cores).

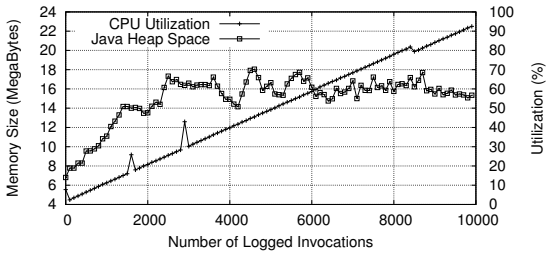


Figure 14: Resource Consumption for Constraint Queries

The increase of time is inherent to the problem of querying growing log data. We argue that query performance is feasible

[7]Note that all processes from Table 3 where implemented and evaluated with the same rigor. However, we do believe that certain parts of our evaluation are best explained in detail based on a single process. Therefore, Sections 7.1 and 7.2 exemplarily discuss the results from the patient examination example. This discussion applies analogously to the other processes from Table 3. The aggregated results for all processes are discussed in Section 7.2.3.

[8]The standard deviation was in the range of 39.21 to 413.69 ms (lowest and highest values are for processes with 1 scope and 18 scopes, respectively) for the secured version, and in the range of 10.38 to 58.78 ms (for 13 scopes and 8 scopes, respectively) for the unsecured version.

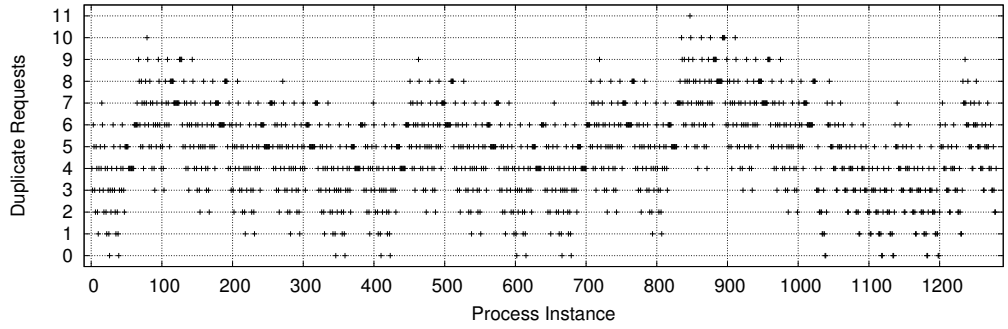


Figure 15: Blocked Task Executions per Test Process Instance (Patient Examination Scenario)

for medium-sized to even large scenarios. Firstly, as evidenced in Figure 14, the execution time appears to grow only linearly (we have also performed a linear regression which showed almost perfect fit for $y = 20 + 0.06x$). The reason is that the queries are formulated in a way that always only the last added log entry needs to be compared to the other entries (hence, the queries are executed for each new log entry). Secondly, even for large logs (tens of thousands of entries) the execution time is still in a range of only a few seconds. If we extrapolate the test values for very huge logs (millions of entries), however, the current approach would take in the order of minutes, which may not be feasible for real-time processes. Hence, additional optimizations will be required for such very-large scale situations – a problem we actively tackle in our future work.

7.2. Reaction of the Secured Process to Valid and Invalid Authentication Data

In the second experiment, we utilize the five evaluation processes (see Section 7) to evaluate how our approach deals with authentication data of authorized and unauthorized users provided by the IdP service. As outlined in Section 4, the task of the IdP is solely to authenticate users, but the authorization in terms of RBAC constraints is enforced by the process instance (and, additionally, by the log data queries from Section 6.4). Hence, the reason for performing this experiment is to test the ability of the transformed business process to cope with unauthorized users who attempt to execute restricted process tasks. Moreover, we are interested in evaluating under which circumstances the RBAC rules may become overconstrained such that the process ends in a deadlock and is unable to continue. Our methodology in this experiment is to execute all possible instances of the test processes with respect to user authorization (given a set of subjects and process tasks, try each combination of subjects performing a task; see Section 7.2.1 for details). The chosen scenario processes have a feasible size to perform this full enumeration. We discuss detailed results based on the patient examination process (P1) in Section 7.2.2, and aggregated results over all five processes (P1-P5) in Section 7.2.3.

7.2.1. Permutation of RBAC Assignments

We define the domain $[T_T \rightarrow (S \times R)]$ of RBAC assignment functions, where T_T is the set of BusinessAction task types, S is the set of subjects and R is the set of roles (cf. Section 3.1). The function defines which authentication data should be used for each task type. We then consider all possible permutations of function assignments in this domain, with the restriction that for each pair $(s, r) \in S \times R$ the subject s is directly associated with role r. To keep the domain small, inherited roles are not considered. For instance, in our scenario the pair (*Bob*,*Physician*) is included, but (*Bob*,*Staff*) is not considered, although *Bob* inherits the role *Staff* through *Physician*. Furthermore, note that SME constraints are checked at design-time when defining a process-related RBAC model. The static correctness rules ensure the consistency of the corresponding RBAC models at any time (see [14]). This means that it is not possible to define an inconsistent RBAC model where, for example, a subject or role possesses the right to execute two SME tasks. The respective RBAC model is then applied to make access decisions and to perform task allocations for all process instances. In other words, because for each process instance the allocation of the respective task instances is based on a consistent process-related RBAC model, it is not necessary to check the fulfillment of SME constraints again at runtime (see also [19]).

For each permutation one process instance has been executed, and the IdP service in the test environment is configured to return the authentication data that correspond to the respective permutation. The IdP keeps track of *getAuthenticationData* requests and registers how many duplicate requests are issued for any task type in each process instance. Recall that a duplicate request is always issued if the IdP provides authentication data of a non-authorized user. Thus, each duplicate *getAuthenticationData* request represents a blocked execution of a restricted task (which is the desired/expected behavior).

The purpose of this experiment setup is to empirically evaluate 1) whether the secured process correctly allows/denies access for valid/invalid provided credentials, respectively, and 2) how the platform deals with unresolvable conflicts (if the process deadlocks due to mutual exclusions). For instance,

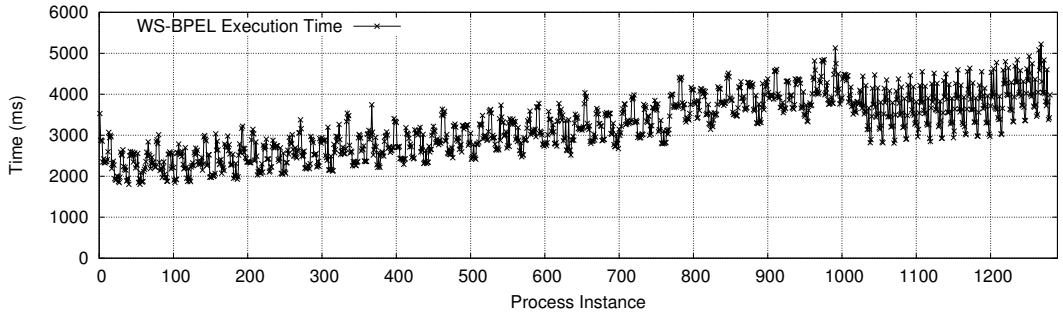


Figure 16: Execution Time of Secured BPEL Process Instances Over Time

when *Get Personal Data* in our scenario has been invoked with (*Bob*,*Physician*) and the IdP provides (*John*,*Staff*) for *Assign Physician*, then it is required to get new authentication data because of a violated role-binding constraint. In this case, the IdP simply provides the next available authentication data, simulating the real-life situation that a new subject logs in after an unauthorized subject has been denied access. This procedure is repeated as long as new pairs of subject and role can be provided; if the process has unsuccessfully attempted to invoke a task with *all* possible combinations, the whole process terminates with a fault message. Note that this method of deadlock detection is suitable for our scenario which defines only a small number of subjects; for more advanced detection of deadlocks and unsatisfiable constraints we refer to related work [44, 45].

7.2.2. Detailed Discussion for the Patient Examination Process

In our scenario, the domain ($S \times R$) consists of the four pairs ((*John,Staff*), (*Jane,Physician*), (*Bob,Physician*), (*Alice,Patient*)), and six task types exist ($|T_T| = 6$). Hence, the total number of possible assignment function permutations is 4^6 =4096. However, the process structure allows to reduce this number because the decision node (whether the patient is in an emergency situation) splits the process into two possible execution paths (one path with 5 tasks and the other path with 4 tasks). The decision node has been simulated to uniformly use both of the two possible conditional branches. Therefore, in total only $4^5 + 4^4$ =1280 process instances have to be executed.

Figure 15 illustrates the number of blocked authorization requests for each process instance. Considering the procedure of security enforcement (cf. Section 4), a blocked request means that the authentication data provided by the IdP violate any constraints (which is expected in many cases, since all permutations are tested). Table 4 summarizes the aggregated values: 20 of the 1280 generated RBAC assignments were completely valid from the start and no blocked requests were necessary. The remaining instances required between 1 and 11 blocked requests until a final state (successful or unsuccessful) is reached.

While there have been 1024 successful executions of the process, 256 failed instances had to be aborted because of deadlock situations. Deadlocks can result from the complex inter-dependencies of BusinessActivity access rules (see, e.g., [18, 46]). For instance, consider the operation sequence in Table 5. The deadlock is caused by the subject-binding between *Get Critical History* and *Decide On Treatment*, combined with the fact that both tasks can be executed by different roles (the former by *Patient* and *Physician*, and the latter only by *Patient*). In fact, all process executions in which the patient *Alice* executes *Get Critical History* lead to this conflicting situation. Note that the focus of this paper is to enforce RBAC constraints and to *detect* deadlocks[9]. In our future work we also investigate techniques to check the satisfiability of a certain process and *avoid* deadlocks in advance (see, e.g., [18, 44, 45, 47]).

Result Outcome	**Instances**	**Result Outcome**	**Instances**
No Blocked Requests	20	7 Blocked Requests	140
1 Blocked Request	56	8 Blocked Requests	80
2 Blocked Requests	108	9 Blocked Requests	32
3 Blocked Requests	163	10 Blocked Requests	10
4 Blocked Requests	228	11 Blocked Requests	1
5 Blocked Requests	232	Successful Execution	1024
6 Blocked Requests	210	Failed (Deadlocked)	256
		Total Instances	1280

Table 4: Process Executions with Permutations of $T_T \rightarrow (S \times R)$ Assignments

Task	**Sub.**	**Role**	**Effect**
Get Personal Data	*John*	*Staff*	Role *Staff* must *Assign Physician* *John* must *Assign Physician*
Assign Physician	*John*	*Staff*	-
Obtain X-Ray Image	*Bob*	*Physician*	-
Get Critical History	*Alice*	*Patient*	*Alice* must not *Get Expert Opinion* *Alice* must *Decide On Treatment*
Get Expert Opinion	*Jane*	*Physician*	-
Decide On Treatment	*?*	*?*	Deadlock, because the bound subject *Alice* is not permitted

Table 5: Operation Sequence Leading to a Constraint Conflict (Deadlock)

[9]Note that the deadlocks in our evaluation result from the fact that we automatically generate and execute all possible process instances (see Section 7.2). Because our process-related RBAC models adhere to the static and dynamic consistency requirements defined in [14, 19] the resulting RBAC models are always consistent. However, even though we always have consistent models, it is still possible that a certain process is not satisfiable (see, e.g., [44, 45]).

The same experiment setup has been used to measure the execution time of the secured process instances over time (Figure 16). Again, we see a slight upwards trend in the processing time. The reasons for this trend are twofold. First, the more instances have executed, the more log data must be checked for constraint conflicts. Second, particularly for SME constraints an increasing number of log data increases the likelihood that the blocked requests need to be issued because the provided test authentication data are in a conflict with one or more previous invocations. The spikes in Figure 16 indicate different execution times of instances with few versus many blocked requests (see also Figure 15). Notice that the execution time shows a certain pattern between roughly 0 and 1000, and a different pattern between 1000 and 1280. These patterns are a direct result of the experiment design, because we first execute 1024 instances that follow the "emergency" path in the scenario process, and afterwards 256 instances that follow the "non-emergency" path.

7.2.3. Aggregated Results for All Test Processes

ID	Instances	Deadlocks	Blocked Requests			Execution Time (ms)		
			min	avg	max	min	avg	max
P1	1280	256	0.0	4.8	11.0	1802.0	3199.6	5222.0
P2	729	243	0.0	3.3	7.0	3990.0	5009.0	8881.0
P3	625	0	0.0	3.6	8.0	3444.0	5464.8	8057.0
P4	3125	0	0.0	6.9	16.0	2984.0	8356.6	14363.0
P5	64	0	0.0	1.8	4.0	2799.0	3070.1	5530.0

Table 6: Aggregated Test Execution Results of the Five Evaluated Processes

Table 6 summarizes the test results for the five test processes. The table contains the process *ID* that refers back to Table 3, the total number of executed instances which were generated from the RBAC assignment permutations, the number of deadlocks that occurred, the blocked requests (minimum/maximum/average) per process instance, and the aggregated execution time per instance. In general, the number of instances corresponds to $|S|^{|T_T|}$, except in cases where we can take advantage of the process structure to reduce the number of instances (i.e., 1280 instead of 4096 instances for P1). Process P4 has the highest number of instances (3125). The aggregated values are computed over all process instances; for example, the average number of blocked requests over all 1280 instances of process P1 is 4.8. The difference between minimum and maximum execution time depends on the executed tasks, and hence correlates strongly with the number of blocked requests. The maxium execution time was roughly 14 seconds (for an instance of process P4), and the shortest instance (of P1) executed within less than 2 seconds. Depending on the process definition and the chosen subjects, either all generated process instances were able to execute successfully (P3, P4, P5), or some instances deadlocked (P1, P2). Some process definitions are prone to deadlocking (e.g., 20% of P1's possible instances lead to a deadlock), whereas in other processes deadlocks are not even possible. For instance, the tax refund process [16] (P4) was run with the smallest possible number of subjects (at least 2 clerks and 3 managers are required), but out of the 3125 instances (each subject tries to access each of the five task types, $5^5 = 3125$) not a single instance deadlocks. Even though satisfiability of access constraints at different points of the process execution can be determined algorithmically (see, e.g., [18]), we argue that it is equally important to test the running system, and to empirically verify the number of successful and blocked requests, as shown in this evaluation.

7.3. WS-BPEL Transformation Algorithm

Concerning the evaluation of the WS-BPEL transformation algorithm, we consider the same twenty test process definitions with different sizes described earlier in Section 7.1.

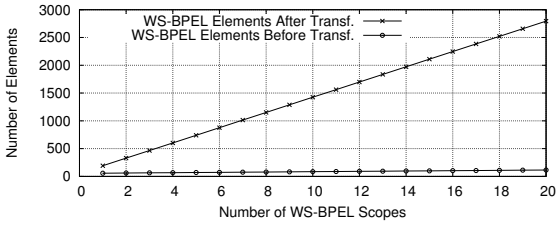


Figure 17: Different Sizes of WS-BPEL Processes Before Transformation (i.e., process annotated with RBAC DSL statements) and After Transformation (i.e., process with injected security enforcement tasks)

Figure 17 shows the number of WS-BPEL elements of the process definition before and after the automatic transformation. The results indicate that the size of the WS-BPEL definition rises with increasing number of scopes. While our test process with a single scope contains 33/115 WS-BPEL elements before/after transformation, the process definition for 10 scopes grows to 60/484 WS-BPEL elements before/after transformation, respectively. These numbers are determined by counting all XML (sub-)elements in the WS-BPEL file using the XPath expression `count(//*)`. At the beginning of the transformation, 41 elements are added (`import`, `partnerLink` and `variable` declarations), and for each new scope 41 elements are added for the IAM task definitions (note that both values are 41 coincidentally). We observe that the ability to define security annotations in WS-BPEL keeps the process definition clear at design time. In fact, the additional code for security enforcement in WS-BPEL is often larger than the actual business logic. This can be seen as an indicator that our approach can reduce the development effort as compared to manual implementation, although we did not empirically evaluate this aspect in detail.

7.4. Limitations

In this section, we discuss the current limitations and weaknesses of our approach and the corresponding Web service technology projection. We also propose possible mechanisms and future work to mitigate the consequences and risks associated with these limitations.

- Parallel Process Flows: WS-BPEL provides the `flow` command for concurrent execution of tasks. Security enforcement of tasks that execute in parallel poses a challenge for various reasons. Firstly, if two tasks are started

with mutually exclusive access rights, a race condition is created with respect to the first task to access the authentication token. Secondly, since we make use of "global" (process-instance-wide) variables, the injected IAM tasks for each single WS-BPEL `invoke` action are supposed to execute atomically and should not access these variables concurrently. To handle parallel execution, we hence propose to extend the injected IAM tasks with two additional tasks to acquire and release an exclusive lock when entering and leaving the critical region, respectively. Since BPEL does not provide a corresponding language construct, an external Web service is used to acquire the exclusive lock on a semaphore. For brevity and clarity, these additional synchronization tasks have not been added to the transformation in Section 4. In future work, we further plan to introduce more sophisticated synchronization using the WS-BPEL `link` mechanism.

- Deadlocking: If the RBAC policies are conflicting, the procedure for obtaining and checking user authentication data can end up in a deadlock that is unable to terminate with a successful result. To mitigate the effect of policy conflicts, it is therefore required to perform timely satisfiability checks. In Section 8 we discuss related work that focuses on this topic, in particular we refer to previous work in [18, 19, 46, 47].

- Single Point of Failure: Our Web service technology projection builds on the assumption that the IdP and Logging services operate reliably and continuously. An outage of any of these services would imply that the access control procedure cannot be performed in its entirety or that certain log data cannot be stored. Depending on the process definition at hand, the consequences can be more or less severe. The IdP service is the key component that provides the basis for user authentication. If it is unavailable, the secured execution fails. A possible strategy for certain application scenarios would be to define *break-the-glass* (BTG) rules (see, e.g., [48, 49, 50]) which allow to temporarily access the protected resources with fallback security settings, in order to provide for continuous operation. An outage of the Logging service is less severe, because it is strictly only required to perform a posteriori conformance checks of global constraints that may affect all (or at least multiple) process instances (see, e.g., [51]). Instance-specific constraints are local to a certain process instance and can be enforced by means of instance-specific log data stored in WS-BPEL variables (see Section 5).

- Security Token Hijacking: Malicious users may attempt to gain access to services they are not entitled to. Consider an attacker who intentionally does not follow the processing logic of the transformed process but invokes the target Web services directly. The attacker may obtain a SAML token by executing *getAuthenticationData*, which asserts its subject and role. Assume that the token is used in combination with the *instanceID* of an active process instance to invoke the *Decide On Treatment*; this situation must be avoided under any circumstances. To enforce the subject-binding with *Get Critical History* and other RBAC rules it is imperative that all access constraints are validated on the service side. In our architecture we hence require a policy enforcement point (PEP) which intercepts and analyzes all invocations.

- Invalid WS-BPEL Modification: For the approach to work reliably, it is important that the WS-BPEL definition should not be modified after the automated code transformation step. We therefore propose the use of a trusted deployment component which provides exclusive access to the business process execution engine. As part of transformation process the WS-BPEL file is signed with an XML signature [52], which is then checked by the deployment component to enforce integrity.

- Human Factors: In the end, a business process involving human labor can only be as safe and reliable as the persons who perform it. That is, control mechanisms such as mutual exclusion (e.g. to enforce the four-eyes principle) can provide a strong instrument for improving quality and reliability, but human errors can never be fully ruled out.

8. Related Work

This section provides a discussion of related work in the area of model-driven IAM and their application to SOA business processes. Our analysis focuses on three main research areas: security modeling for Web service based systems, DSL-based security modeling, and techniques for incorporating runtime enforcement of security constraints into business processes.

8.1. Security Modeling for Web Service Based Systems

Jensen and Feja [53] discuss security modeling of Web service based business processes, focusing on access control, confidentiality and integrity. Their approach is based on Event-driven Process Chains (EPC) [54] and defines different security symbols that the process definitions are annotated with. Their implementation is integrated into the ARIS SOA Architect software, which is also able to transform the EPC model into an executable SOA business process. The paper describes the generation of WS-SecurityPolicy [55] policies, but does not discuss mutual exclusion and binding constraints in process-related RBAC models, nor does it discuss in detail how the process engine enforces the policies and constraints at runtime, which in contrast is a core part in our work.

Kulkarni et al. [56] describe an application of context-aware RBAC to pervasive computing systems. As the paper rightly states, model-level support for revocation of roles and permissions is required to deal with changing context information. The approach has a strong focus on dynamically changing context (e.g., conditions measured by sensors) and the associated permission (de-)activation. In our framework, context information is part of the RBAC model definitions (more details can be found in [21]). In this paper, the context information in the RBAC model has been abstracted from, but as part of our future

work we plan to integrate the Business Activity model in [14] with context information (see also [57]).

Although our model does not directly build on the notion of trust, access control policies can also be established dynamically by deriving trust relationships among system participants [58]. Skoksrud et al. present Trust-Serv [59], a solution for model-driven trust negotiation in Web service environments. Similar to our approach, the policy enforcement is transparent to the involved Web services. Another similarity is that trust credentials (such as user identifier, address or credit card number) are exchanged iteratively throughout the process, which is also the case for the authentication credentials in our approach. However, trust-based policies in [59] are *monotonic* in the sense that additional trust credentials always add access rights and never remove existing ones, which is in contrast to access control in this paper, where the execution of tasks can activate entailment constraints which progressively narrow down the set of valid access control configurations.

Our approach was also influenced by Foster et al. [60] who present an integrated workbench for model-based engineering of service compositions. Their approach supports service and business process developers by applying formal semantics to service behavior and configuration descriptors, which can then be analyzed and checked by a verification and validation component. The policies enforced by the workbench are quite generally applicable and hence require developers to perform application specific modeling, whereas our proposed DSL and WS-BPEL annotations are tailored to the domain of RBAC and entailment constraints and arguably straight-forward to apply.

Seminal contributions in the context of modeling support for Web service based business processes are provided within the Web Services Modeling Framework (WSMF) by Fensel et al. [61], and the modeling ontologies that emerged from this project. For instance, security requirements can be modeled in WSMF by declaring the subject and role as input data and defining pre-conditions for all operations that require certain authentication data. In the previous years, the Semantic Web community has been pushing forward various ontologies to draw an ever more exact picture of the functionality exposed by Web services, in order to allow for sophisticated discovery, execution, composition and interoperation [62]. In fact, although not very frequently used in practice, semantically annotated Web services also allow for a more fine-grained definition of access control policies, from the interaction level down to the message level. Whereas annotations in semantic Web services are used mostly for reasoning purposes, the BPEL annotations used in our approach are utilized as metadata for runtime access control enforcement. Such business process model abstractions, which are the underpinning of semantic equivalence and structural difference, have been empirically studied in [63], and our approach can be seen as the reverse operation of abstraction (i.e., concretization) for the specific application domain of task-based entailment constraints.

Various other papers have been published that are related to our work or have influenced it, some of which are mentioned in the following. The platform-independent framework for Security Services named SECTISSIMO has been proposed by Memon at al. [64]. The conceptual novelty of this framework is the three-layered architecture which introduces an additional layer of abstraction between the models and the concrete implementation technologies. In contrast, our prototype only considers two layers (i.e. modeling of RBAC constraints and transformation of WS-BPEL code). However, the presented modeling concepts (see Section 3) as well as the model transformations (see Section 4) are independent from concrete implementation technologies too.

Lin et al. [65] propose a policy decomposition approach. The main idea is to decompose a global policy and distribute it to each collaborating party. This ensures autonomy and confidentiality of each party. Their work is particularly of relevance for cross-organizational definition of RBAC policies, as performed in our multi-hospital use case scenario. Currently, our prototypical implementation relies on a single, global RBAC Web service. However, we plan to adopt this complementary policy decomposition approach, which will allow each hospital to employ its own dedicated RBAC Web service.

8.2. DSL-Based Security Modeling

An integrated approach for Model Driven Security, that promotes the use of Model Driven Architectures in the context of access control, is presented by Basin et al. [66]. The foundation is a generic schema that allows creation of DSLs for modeling of access control requirements. The domain expert then defines models of security requirements using these languages. With the help of generators these models are then transformed to access control infrastructures. However, compared to our approach, [66] does not address the definition of task-based entailment constraints.

The approach by Wolter et al. [36] is concerned with modeling and enforcing security goals in the context of SOA business processes. Similar to our approach, their work suggests that business process experts should collaboratively work on the security policies. They define platform independent models (PIM) which are mapped to platform specific models (PSM). At the PIM level, XACML and *AXIS 2*[10] security configurations are generated. Whereas their approach attempts to cover diverse security goals including integrity, availability and audit, we focus on entailment constraints in service-based business processes.

A related access control framework for WS-BPEL is presented by Paci et al. in [67]. It introduces the *RBAC-WS-BPEL* model and the authorization constraint language *BPCL*. Similar to our approach, the BPEL activities are associated with required permissions (in particular, we associate permissions for `invoke` activities that try to call certain service operations). However, one main difference is related to the boundaries of the validity of user permissions: RBAC-WS-BPEL considers pairs of adjacent activities (a_1 and a_2, where a_1 has a control flow link to a_2) and defines rules among them, including separation of duty (a_1 and a_2 must execute under different roles) and binding of duty (a_1 and a_2 require the same role or user). As

[10] http://axis.apache.org/axis2/java/core/

elaborated in previous work [21], our approach also allows to annotate scopes (groups of `invoke` tasks) in BPEL processes and hence to apply RBAC policies in a sequential, but also in a hierarchical manner.

XACML [68] is an XML-based standard to describe RBAC policies in a flexible and extensible way. Our DSL could be classified as a high-level abstraction that implements a subset of XACML's feature set. Using a transformation of DSL code to XACML markup, it becomes possible to integrate our approach with the well-established XACML environment and tools for policy integration (e.g., [69]).

8.3. Runtime Enforcement of Security and Other Constraints in Business Processes

Various approaches have been proposed to incorporate extensions and cross-cutting concerns such as security features into business process models. Most notably, we can distinguish different variants of model transformation [70, 30] and approaches that use aspect-oriented programming [71].

A dynamic approach for enforcement of Web services Security is presented in [72] by Mourad et al. The novelty of the approach is mainly grounded by the use of Aspect-Oriented Programming (AOP) in this context, whereby security enforcement activities are specified as *aspects* that are dynamically woven into WS-BPEL processes at certain *join points*. Charfi and Mezini presented the AO4BPEL [73] framework, an aspect-oriented extension to BPEL that allows to attach cross-cutting concerns. The aspect-oriented language Aspects for Access Control (AAC) by Braga [74] is based on the same principle and is capable of transforming SecureUML [75] models into aspects. A main difference is that AAC does not operate on BPEL, but on Java programs, and can hence be applied directly to Java Web service implementations to enforce access control.

Essentially, our approach can be regarded as a variant of AOP: the weaved aspects are the injected IAM tasks, and join points are defined by security annotations in the process. A major advantage of our approach is the built-in support for SSO and cross-organizational IAM. An interesting extension could be to decouple security annotations from the WS-BPEL definition, to store them in a separate repository and to dynamically adapt to changes at runtime.

A plethora of work has been published on transformations and structural mappings of business process models. Most notably, our solution builds on work by Saquid/Orlowska [76], and Eder/Gruber [77] who presented a meta model for block structured workflow models that is capable of capturing atomic transformation actions. These transformation building blocks are important for more complex transformations, as in our case when multiple process fragments for enforcement of entailment constraints are combined for a single action in WS-BPEL. While this work focuses mainly on deployment time model transformations, other research also investigates runtime changes of service compositions. For instance, automatic process instrumentation and runtime transformation have previously been applied in the context of functional testing [78] of service-based business processes. Weber et al. [79] investigate security issues in adaptive process management systems and claim that such dynamicity increases the vulnerability to misuse. Our approach is adaptive in that it allows the "environment" (e.g., access policies) to change at runtime. However, we currently assume that the process definition itself does not change. In our ongoing research, we are complementing our approach with support for online structural process adaptation.

An important aspect of security enforcement is the way how constraint conflicts are handled at runtime. Consequently, our approach is related to a recent study on handling conflicts of binding and mutual exclusion constraints in business processes [46, 47]. Based on a formalization of process-related RBAC, this work proposes algorithms to detect conflicts in constraint definitions, as well as strategies to resolve the conflicts that have been detected. In our evaluation (see Section 7), we illustrated an example constraint conflict that lead to a deadlock and discussed how the platform is able to detect such conflicts. In order to anticipate and avoid deadlocks altogether, we will eventually integrate these algorithms with our RBAC DSL.

Although not necessarily concerned with security (i.e., access control) in the narrower sense, the area of Web service transaction processing [80, 81] and conversational service protocols [82, 83] is related to our work on secured business processes. Put simply, a transactional protocol is a sequence of operations with multiple participants that have a clearly defined role and need to collaboratively perform a certain task. Analogously, BusinessActivities are performed by subjects with clearly defined roles and limited permissions. One could argue that while the responsibility of transaction control is to ensure that all participants actually *do* perform their task, the main purpose of access control is to ensure that subjects *do not* perform tasks they are not authorized to. Amongst others, our approach was influenced by von Riegen et al. [81] who model distributed Web service transactions with particular focus on complex interactions where participants are restricted to only possess limited local views on the overall process. These limited views are comparable to our access control enforcement. Our approach also detects if a process instance is about to break the required conversational protocol (i.e., access control policies), in which case we apply a sequence of compensation actions [80] (e.g., repeat authentication or terminate instance due to deadlock).

9. Conclusion

We presented an integrated, model-driven approach for the enforcement of access control policies and task-based entailment constraints in distributed service-based business processes. The approach is centered around the DSL-driven development of RBAC policies and the runtime enforcement of the resulting policies and constraints in Web services based business processes. Our work fosters cross-organizational authentication and authorization in service-based systems, and facilitates the systematic development of secured business processes. From the modeling perspective, the solution builds on the BusinessActivity extension – a native UML extension for defining entailment constraints in activity diagrams. We provided a detailed description of the procedure to transform design-time

BusinessActivity models into standard activity models that enforce the access constraints at runtime. Based on a generic transformation procedure, we discussed our implementation which is based on WS-BPEL and the Web services framework.

Our approach based on BusinessActivities allows to abstract from the technical implementation of security enforcement in the design time view of process models. The detailed evaluation of the process transformation has shown that process definitions with injected tasks for security enforcement grow considerably large. In fact, the additional code for security enforcement in WS-BPEL is often larger than the actual business logic. This can be seen as an indicator that our approach can reduce the development effort as compared to manual implementation, although we did not empirically evaluate this aspect in detail.

Our extensive performance evaluation has illustrated that the proposed runtime enforcement procedures operate with a slight overhead that scales well up to the order of several ten thousand logged invocations. We can conclude that the overhead consists of three main parts: 1) the approach builds on digital signatures for ensuring message integrity, 2) the process determines the role and permissions of the currently executing user, which results in additional requests and increased execution time, and 3) the enforcement of entailment constraints requires querying the log traces of previous executions of the process. Note that the overhead for 1) and 2) does not increase over time (with rising number of process executions), whereas the overhead for 3) inherently rises because the log traces are accumulating over time, and more data have to be evaluated.

The implementation of our prototype still has limitations, and we discussed strategies to improve some of these limitations in future work. For instance, advanced synchronization mechanisms are required for business processes with highly parallel processing logic. Moreover, the query mechanism that checks security constraints for validity needs to be further optimized for very large log data sets (in the order of millions of invocations). We envision advanced data storage and compression techniques, as well as optimized query mechanisms to further reduce this increase of overhead over time. In our ongoing work we also investigate the use of additional security annotations and an extended view of context information. Finally, we plan to shift from a process-centric to a more data-centric view and integrate the concept of entailment constraints to our recent work on reliability in event-based data processing [84] and collaborative Web applications [85].

Acknowledgements

This work is partially supported by the Austrian Science Fund (FWF): P23313-N23, and has received funding from the European Community's Seventh Framework Programme (FP7) under grant agreement 257483 (Indenica).

Appendix A. RBAC DSL Statements for Scenario Process

Listing 5 contains the complete access control configuration of the Patient Examination scenario process (two involved hospitals), expressed using RBAC DSL statements.

```
1  RESOURCE   PatientService1 "http://hospital1.com/patients"
2  RESOURCE   PatientService2 "http://hospital2.com/patients"
3  OPERATION  retrieveData
4  OPERATION  makeAssignment
5  OPERATION  getHistory
6  OPERATION  getOpinion
7  OPERATION  queryPartner
8  OPERATION  makeDecision
9  ROLE       Staff
10 ROLE       Physician
11 ROLE       Patient
12 INHERIT    Staff Physician
13 SUBJECT    John
14 SUBJECT    Jane
15 SUBJECT    Bob
16 SUBJECT    Alice
17 ASSIGN     John Staff
18 ASSIGN     Jane Physician
19 ASSIGN     Bob Physician
20 ASSIGN     Alice Patient
21  # Web service operation permissions (hospital 1)
22 PERMIT     Staff      retrieveData    PatientService1
23 PERMIT     Staff      makeAssignment  PatientService1
24 PERMIT     Physician  getHistory      PatientService1
25 PERMIT     Patient    getHistory      PatientService1
26 PERMIT     Physician  getOpinion      PatientService1
27 PERMIT     Patient    queryPartner    PatientService1
28 PERMIT     Physician  makeDecision    PatientService1
29  # Web service operation permissions (hospital 2)
30 PERMIT     Staff      retrieveData    PatientService2
31 PERMIT     Staff      makeAssignment  PatientService2
32 PERMIT     Physician  getHistory      PatientService2
33 PERMIT     Patient    getHistory      PatientService2
34 PERMIT     Physician  getOpinion      PatientService2
35 PERMIT     Patient    queryPartner    PatientService2
36 PERMIT     Physician  makeDecision    PatientService2
37  # 'task' to 'service operation' bindings (hospital 1)
38 TASK       GetPersonalData retrieveData    PatientService1
39 TASK       AssignPhysician makeAssignment  PatientService1
40 TASK       GetCriticalHistory getHistory   PatientService1
41 TASK       GetExpertOpinion getOpinion     PatientService1
42 TASK       GetPartnerHistory queryPartner  PatientService1
43 TASK       DecideOnTreatment makeDecision  PatientService1
44  # 'task' to 'service operation' bindings (hospital 2)
45 TASK       GetPersonalData retrieveData    PatientService2
46 TASK       AssignPhysician makeAssignment  PatientService2
47 TASK       GetCriticalHistory getHistory   PatientService2
48 TASK       GetExpertOpinion getOpinion     PatientService2
49 TASK       GetPartnerHistory queryPartner  PatientService2
50 TASK       DecideOnTreatment makeDecision  PatientService2
51  # task-based entailment constraints
52 RBIND      GetPersonalData AssignPhysician
53 DME        GetCriticalHistory GetExpertOpinion
54 SBIND      GetCriticalHistory DecideOnTreatment
55 SBIND      GetPartnerHistory GetPartnerHistory
56 SME        GetExpertOpinion GetPartnerHistory
```

Listing 5: Exemplary RBAC DSL Statements for Hospital Scenario

References

[1] D. Draheim, The Service-Oriented Metaphor Deciphered, JCSE 4 (4) (2010) 253–275.

[2] M. Huhns, M. Singh, Service-Oriented Computing: Key Concepts and Principles, IEEE Internet Computing 9 (2005) 75–81.

[3] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, Service-Oriented Computing: State of the Art and Research Challenges, IEEE Computer 40 (11) (2007) 38–45.

[4] World Wide Web Consortium (W3C), Web services activity. URL http://www.w3.org/2002/ws/

[5] P. Leitner, W. Hummer, S. Dustdar, Cost-Based Optimization of Service Compositions, IEEE Transactions on Services Computing (Preprint) (99) (2011) 1.

[6] J. Cannon, M. Byers, Compliance Deconstructed, ACM Queue 4 (7) (2006) 30–37.

[7] M. Damianides, How does SOX change IT?, Journal of Corporate Accounting & Finance 15 (6) (2004) 35–41.

[8] S. Mishra, H. Weistroffer, A Framework for Integrating Sarbanes-Oxley Compliance into the Systems Development Process, Communications of the Association for Information Systems (CAIS) 20 (1) (2007) 712–727.

[9] D. F. Ferraiolo, D. R. Kuhn, R. Chandramouli, Role-Based Access Control, 2nd Edition, Artech House, 2007.

[10] R. Sandhu, E. Coyne, H. Feinstein, C. Youman, Role-based access control models, Computer 29 (2) (1996) 38–47.

[11] M. Strembeck, A Role Engineering Tool for Role-Based Access Control, in: 3rd Symposium on Requirements Engineering for Information Security, 2005.

[12] M. Strembeck, Scenario-driven Role Engineering, IEEE Security & Privacy 8 (1) (2010) 28–35.

[13] J. Wainer, P. Barthelmes, A. Kumar, W-RBAC - A Workflow Security Model Incorporating Controlled Overriding of Constraints, International Journal of Cooperative Information Systems 12 (4) (2003) 455–485.

[14] M. Strembeck, J. Mendling, Modeling Process-related RBAC Models with Extended UML Activity Models, Information and Software Technology 53 (5) (2011) 456–483.

[15] D. Clark, D. Wilson, A Comparison of Commercial and Military Computer Security Policies, in: IEEE Symp. on Security and Privacy, 1987.

[16] E. Bertino, E. Ferraria, V. Atluri, The specification and enforcement of authorization constraints in workflow management systems, ACM Transactions on Information and System Security 2 (1) (1999) 65–104.

[17] R. Botha, J. Eloff, Separation of duties for access control enforcement in workflow environments, IBM Systems Journal 40 (3) (2001) 666–682.

[18] K. Tan, J. Crampton, C. Gunter, The Consistency of Task-Based Authorization Constraints in Workflow Systems, in: 17th IEEE Workshop on Computer Security Foundations (CSFW), 2004, pp. 155–169.

[19] M. Strembeck, J. Mendling, Generic Algorithms for Consistency Checking of Mutual-Exclusion and Binding Constraints in a Business Process Context, in: 18th International Conference on Cooperative Information Systems (CoopIS), 2010.

[20] C. Wolter, A. Schaad, C. Meinel, Task-based entailment constraints for basic workflow patterns, in: 13th ACM Symposium on Access Control Models and Technologies (SACMAT), ACM, 2008, pp. 51–60.

[21] W. Hummer, P. Gaubatz, M. Strembeck, U. Zdun, S. Dustdar, An integrated approach for identity and access management in a SOA context, in: 16th ACM Symposium on Access Control Models and Technologies (SACMAT), 2011, pp. 21–30.

[22] D. C. Schmidt, Guest editor's introduction: Model-driven engineering, IEEE Computer 39 (2) (2006) 25–31.

[23] B. Selic, The Pragmatics of Model-Driven Development, IEEE Software 20 (5) (2003) 19–25.

[24] T. Stahl, M. Völter, Model-Driven Software Development, John Wiley & Sons, 2006.

[25] M. Mernik, J. Heering, A. Sloane, When and How to Develop Domain-Specific Languages, ACM Computing Surveys 37 (4) (2005) 316–344.

[26] D. Spinellis, Notable design patterns for domain-specific languages, Journal of Systems and Software 56 (1) (2001) 91–99.

[27] U. Zdun, M. Strembeck, Reusable Architectural Decisions for DSL Design: Foundational Decisions in DSL Projects, in: 14th European Conference on Pattern Languages of Programs (EuroPLoP), 2009.

[28] M. Strembeck, U. Zdun, An Approach for the Systematic Development of Domain-Specific Languages, Software: Practice and Experience 39 (15).

[29] T. Mens, P. V. Gorp, A Taxonomy of Model Transformation, Electronic Notes in Theoretical Computer Science 152 (2006) 125–142.

[30] S. Sendall, W. Kozaczynski, Model Transformation: The Heart and Soul of Model-Driven Software Development, IEEE Software 20 (5).

[31] U. Zdun, M. Strembeck, Modeling Composition in Dynamic Programming Environments with Model Transformations, in: 5th Int. Symposium on Software Composition, 2006.

[32] Object Management Group, UML 2.4.1 Superstructure (August 2011).
URL http://www.omg.org/spec/UML/2.4.1

[33] OASIS, Web Services Business Process Execution Language (2007).
URL http://docs.oasis-open.org/wsbpel/2.0/OS

[34] OASIS, Security Assertion Markup Language (March 2005).
URL http://docs.oasis-open.org/security/saml-/v2.0/saml-core-2.0-os.pdf

[35] World Wide Web Consortium (W3C), SOAP Messaging Framework (2007).
URL http://www.w3.org/TR/soap12-part1/

[36] C. Wolter, M. Menzel, A. Schaad, P. Miseldine, C. Meinel, Model-driven business process security requirement specification, Journal of Systems Architecture 55 (2009) 211–223.

[37] Object Management Group, Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification (January 2011).
URL http://www.omg.org/spec/QVT/

[38] OASIS, Metadata for the OASIS Security Assertion Markup Language (SAML) (2005).
URL http://docs.oasis-open.org/security/saml-/v2.0/saml-metadata-2.0-os.pdf

[39] World Wide Web Consortium (W3C), XSL Transformations (XSLT) Version 2.0 (2007).
URL http://www.w3.org/TR/xslt20/

[40] B. Hoisl, M. Strembeck, A UML Extension for the Model-driven Specification of Audit Rules, in: 2nd International Workshop on Information Systems Security Engineering (WISSE), Springer Verlag, 2012.

[41] World Wide Web Consortium (W3C), XQuery 3.0: An XML Query Language (2011).
URL http://www.w3.org/TR/xquery-30/

[42] W. Hummer, P. Leitner, S. Dustdar, WS-Aggregation: Distributed Aggregation of Web Services Data, in: ACM Symposium On Applied Computing, 2011.

[43] J. Crampton, A reference monitor for workflow systems with constrained task execution, in: 10th ACM Symposium on Access Control Models and Technologies (SACMAT), 2005, pp. 38–47.

[44] J. Crampton, G. Gutin, A. Yeo, On the parameterized complexity of the workflow satisfiability problem, in: 19th ACM Conference on Computer and Communications Security (CCS), ACM, 2012, pp. 857–868.

[45] Q. Wang, N. Li, Satisfiability and resiliency in workflow authorization systems, ACM Transactions on Information and System Security (TISSEC) 13 (4) (2010) 40:1–40:35.

[46] S. Schefer, M. Strembeck, J. Mendling, A. Baumgrass, Detecting and resolving conflicts of mutual-exclusion and binding constraints in a business process context, in: 19th International Conference on Cooperative Information Systems (CoopIS'11), Springer, 2011.

[47] S. Schefer, M. Strembeck, J. Mendling, Checking satisfiability aspects of binding constraints in a business process context, in: Workshop on Workflow Security Audit and Certification (WfSAC), Springer, 2011.

[48] A. Ferreira, R. Cruz-Correia, L. Antunes, P. Farinha, E. Oliveira-Palhares, D. Chadwick, A. Costa-Pereira, How to break access control in a controlled manner, in: Computer-Based Medical Systems, 2006. CBMS 2006. 19th IEEE International Symposium on, 2006, pp. 847–854.

[49] S. Marinovic, R. Craven, J. Ma, N. Dulay, Rumpole: a flexible break-glass access control model, in: 16th ACM Symposium on Access Control Models and Technologies (SACMAT), 2011.

[50] S. Schefer-Wenzl, M. Strembeck, A UML Extension for Modeling Break-Glass Policies, in: 5th International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA), 2012.

[51] A. Baumgrass, T. Baier, J. Mendling, M. Strembeck, Conformance Checking of RBAC Policies in Process-Aware Information Systems, in: BPM'11 Workshop on Workflow Security Audit and Certification (WfSAC), Springer, 2011.

[52] World Wide Web Consortium (W3C), XML Signature Syntax and Processing (2008).
URL http://www.w3.org/TR/xmldsig-core/

[53] M. Jensen, S. Feja, A security modeling approach for web-service-based business processes, in: 16th Annual IEEE International Conference on the Engineering of Computer Based Systems (ECBS'09), 2009, pp. 340–347.

[54] A.-W. Scheer, O. Thomas, O. Adam, Process Modeling using Event-Driven Process Chains, John Wiley & Sons, Inc., 2005, pp. 119–145.

[55] OASIS, WS-SecurityPolicy 1.3 (2009).
URL http://docs.oasis-open.org/ws-sx/ws-security-policy/v1.3/os/

[56] D. Kulkarni, A. Tripathi, Context-aware role-based access control in pervasive computing systems, in: 13th ACM SACMAT, 2008, pp. 113–122.

[57] M. Strembeck, G. Neumann, An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments, ACM Trans. on Inf. and System Security 7 (3) (2004) 392–427.

[58] N. Dimmock, A. Belokosztolszki, D. Eyers, J. Bacon, K. Moody, Using trust and risk in role-based access control policies, in: 9th ACM Symposium on Access Control Models and Technologies (SACMAT), 2004.

[59] H. Skogsrud, B. Benatallah, F. Casati, Model-Driven Trust Negotiation for Web Services, IEEE Internet Computing 7 (2003) 45–52.

[60] H. Foster, S. Uchitel, J. Magee, J. Kramer, An integrated workbench for model-based engineering of service compositions, IEEE Transactions on Services Computing 3 (2) (2010) 131–144.

[61] D. Fensel, C. Bussler, The web service modeling framework wsmf, Electronic Commerce Research and Applications 1 (2) (2002) 113 – 137.

[62] S. McIlraith, T. Son, H. Zeng, Semantic web services, IEEE Intelligent Systems 16 (2) (2001) 46–53.

[63] S. Smirnov, H. A. Reijers, M. Weske, A semantic approach for business process model abstraction, in: 23rd International Conference on Advanced information Systems engineering (CAiSE), 2011, pp. 497–511.

[64] M. Memon, M. Hafner, R. Breu, SECTISSIMO: A Platform-independent Framework for Security Services, in: Modeling Security Workshop at MODELS '08, 2008.

[65] D. Lin, P. Rao, E. Bertino, N. Li, J. Lobo, Policy decomposition for collaborative access control, in: 13th ACM SACMAT, 2008, pp. 103–112.

[66] D. Basin, J. Doser, T. Lodderstedt, Model driven security: From UML models to access control infrastructures, ACM Transactions on Software Engineering Methodology 15 (2006) 39–91.

[67] F. Paci, E. Bertino, J. Crampton, An Access-Control Framework for WS-BPEL, Int. Journal of Web Services Research 5 (3) (2008) 20–43.

[68] OASIS, eXtensible Access Control Markup Language (2005).
URL `http://docs.oasis-open.org/xacml/2.0`

[69] P. Mazzoleni, B. Crispo, S. Sivasubramanian, E. Bertino, XACML Policy Integration Algorithms, ACM Transactions on Information System Security 11 (2008) 4:1–4:29.

[70] K. Czarnecki, S. Helsen, Feature-based survey of model transformation approaches, IBM Systems Journal - Model-driven software development 45 (2006) 621–645.

[71] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, J. Irwin, Aspect-oriented programming, in: European Conference on Object-Oriented Programming (ECOOP'97), 1997, pp. 220–242.

[72] A. Mourad, S. Ayoubi, H. Yahyaoui, H. Otrok, New approach for the dynamic enforcement of Web services security, in: 8th International Conference on Privacy Security and Trust, 2010, pp. 189–196.

[73] A. Charfi, M. Mezini, AO4BPEL: An Aspect-oriented Extension to BPEL, World Wide Web Journal - Special Issue: Recent Advances in Web Services 10 (2007) 309–344.

[74] C. Braga, A transformation contract to generate aspects from access control policies, Software and Systems Modeling 10 (2011) 395–409.

[75] T. Lodderstedt, D. A. Basin, J. Doser, Secureuml: A uml-based modeling language for model-driven security, in: 5th International Conference on The Unified Modeling Language (UML'02), Springer-Verlag, 2002, pp. 426–441.

[76] W. Sadiq, M. Orlowska, On business process model transformations, in: A. Laender, S. Liddle, V. Storey (Eds.), Conceptual Modeling ER 2000, Vol. 1920, Springer Berlin / Heidelberg, 2000, pp. 47–104.

[77] J. Eder, W. Gruber, A meta model for structured workflows supporting workflow transformations, in: 6th East European Conference on Advances in Databases and Information Systems (ADBIS'02), Springer-Verlag, 2002, pp. 326–339.

[78] W. Hummer, O. Raz, O. Shehory, P. Leitner, S. Dustdar, Test coverage of data-centric dynamic compositions in service-based systems, in: 4th International Conference on Software Testing, Verification and Validation (ICST), 2011.

[79] B. Weber, M. Reichert, W. Wild, S. Rinderle, Balancing flexibility and security in adaptive process management systems, in: On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, Vol. 3760, Springer Berlin / Heidelberg, 2005, pp. 59–76.

[80] M. Schäfer, P. Dolog, W. Nejdl, An environment for flexible advanced compensations of web service transactions, ACM Transactions on the Web 2 (2) (2008) 14:1–14:36.

[81] M. von Riegen, M. Husemann, S. Fink, N. Ritter, Rule-based coordination of distributed web service transactions, IEEE Transactions on Services Computing 3 (1) (2010) 60–72.

[82] B. Benatallah, F. Casati, F. Toumani, Web Service Conversation Modeling: A Cornerstone for E-Business Automation, IEEE Internet Computing 8 (1) (2004) 46–54.

[83] W. Hummer, P. Leitner, S. Dustdar, SEPL – a domain-specific language and execution environment for protocols of stateful Web services, Distributed and Parallel Databases 29 (4) (2011) 277–307.

[84] W. Hummer, C. Inzinger, P. Leitner, B. Satzger, S. Dustdar, Deriving a unified fault taxonomy for event-based systems, in: 6th ACM International Conference on Distributed Event-Based Systems (DEBS'12), 2012.

[85] P. Gaubatz, U. Zdun, Supporting entailment constraints in the context of collaborative web applications, in: 28th Symposium On Applied Computing (SAC), ACM, 2013.

Paper C

Consistency Checking and Resolution Strategies for Runtime Conflicts Resulting from Changes in Process-related RBAC Models

The subsequent paper has been submitted as follows:

T. Quirchmayr, P. Gaubatz, M. Strembeck, and U. Zdun. Consistency Checking and Resolution Strategies for Runtime Conflicts Resulting from Changes in Process-related RBAC Models. submitted to *Advances in Verifiably Secure Process-aware Information Systems*, submitted in June 2014.

In this paper, we studied the fact that changes in process-related RBAC models may have potentially negative impacts on the corresponding process instances. In particular, RBAC policies and constraints which are defined at design time are enforced in runtime process instances. Changing these policies and constraints while process instances are currently running, may inevitably lead to a number of different runtime consistency conflicts. To tackle this problem, we identified and named the different types of such runtime consistency conflicts and introduce novel strategies that allow for the systematic resolution of these conflicts. Our approach is founded on the *Business Activity* meta-model [88] and can therefore considered to be an orthogonal complement of the work presented in Paper A and Paper B.

Consistency Checking and Resolution Strategies for Runtime Conflicts Resulting from Changes in Process-related RBAC Models

Thomas Quirchmayr[1], Patrick Gaubatz[2], Mark Strembeck[1], and Uwe Zdun[2]

[1] Institute for Information Systems, New Media Lab
WU Vienna, Austria
{firstname.lastname}@wu.ac.at
[2] Faculty of Computer Science
University of Vienna, Austria
{firstname.lastname}@univie.ac.at

Abstract. A process-related role-based access control (RBAC) model defines which subjects and roles are allowed to execute the tasks that are included in the business processes of a specific organization. In this context, entailment constraints, such as mutual exclusion and binding constraints, are an additional means to control the task flow in process instances. Changes in the process-related RBAC model may have a direct impact on the respective process instances. In particular, the access control policies and constraints which are defined at design time are enforced in runtime process instances. If these policies and constraints change while a process instance is running, such a change may therefore result in a number of runtime consistency conflicts. In this paper, we discuss different types of such runtime consistency conflicts and introduce corresponding strategies that support the systematic resolution of these conflicts.

Key words: process-aware information systems, task-based entailment constraints, role-based access control, binding of duty, mutual exclusion

1 Introduction

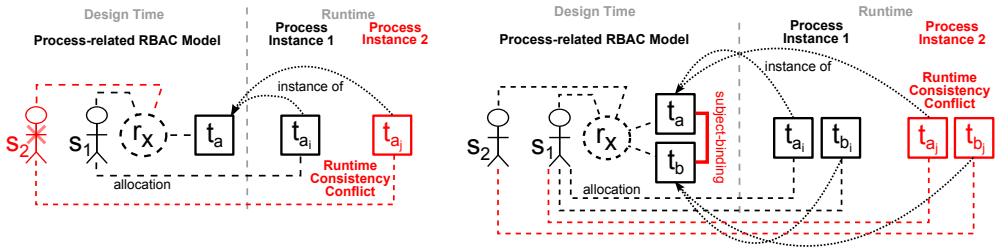
Process-aware information systems (PAIS) [6] enable the execution, management and documentation of business processes in a comprehensive way. A task is a logical unit of work that is connected to and performed via a software service at runtime (e.g., Web services or applications).

Role-based access control (RBAC) is a widely implemented approach to enforce access control (see, e.g., [7, 12]). In the context of workflows, process-related RBAC models define access control policies (see, e.g. [1, 15, 17, 20]) which assign subjects to roles which hold permissions to execute particular tasks. Furthermore, an RBAC mechanism usually specifies authorization constraints, such as mutual exclusion, or binding constraints.

The immanent complexity of process-related RBAC models and the corresponding task-based mutual exclusion constraints and binding constraints require checks to ensure the consistency of the corresponding process-related RBAC model at design time (type-level) and at runtime (instance-level). In our previous work, we identified consistency requirements for entailment constraints and provided generic algorithms to ensure their consistency in a business process context at design-time and runtime (see [14, 15, 16]).

This paper discusses the impact that changes in process-related RBAC models (e.g., adding or removing subjects, roles, or task types) might have on the corresponding process instances. Our considerations are complementary to satisfiability approaches (see, e.g., [3, 5]) as we assume the business process to be satisfiable and the corresponding process-related RBAC model to be consistent before and after a change (see also [14, 15, 16]). In this paper, we identify runtime consistency conflicts in process instances that may result from changes in a corresponding process-related RBAC model. Furthermore, we present generic algorithms to check the consistency at runtime and provide resolution strategies for each runtime consistency conflict.

Figure 1 shows two situations where a change operation in a process-related RBAC model leads to a runtime consistency conflict. In the context of this paper, we define a runtime consistency conflict as a situation where an instance of a business process, that is satisfiable at design time, cannot be executed (completely) as a result of a change of the corresponding process-related RBAC model.



(a) Delete Subject (b) Add Subject-Binding Constraint

Fig. 1. Runtime Consistency Conflicts as a Result of Changes in a Process-Related RBAC Model

In Figure 1(a) subject s_2 is to be deleted from the process-related RBAC model. This change operation is allowed on type level, as task t_a can also be executed by subject s_1 due to its assignment to role r which occupies the permissions to execute t_a. However, as task instance t_{a_j} (i.e., an instance of task t) in *process instance 2* is allocated to s_2 at runtime this inevitably leads to a runtime consistency conflict.

Figure 1(b) shows that adding a subject-binding constraint can also lead to a runtime consistency conflict. At the type level the introduction of a new subject

binding between the two tasks t_a and t_b is allowed (e.g., subjects s_1 and s_2 are allowed to execute both subject-bound tasks t_a and t_b). *Process instance 1* can be executed correctly as task instances t_{a_i} and t_{b_i} are allocated to s_1. In *process instance 2* t_{a_j} is allocated to s_1 whereas t_{b_j} is allocated to s_2 which violates the new introduced subject-binding constraint to be added and thus, leads to a runtime consistency conflict.

In this paper:

- We systematically analyze each change operation in process-related RBAC models regarding their impact on the runtime consistency of corresponding process instances.
- We provide generic algorithms for detecting runtime consistency conflicts for a given change operation.
- We propose (semi-) automatic resolution strategies for each runtime consistency conflict.

The remainder of this paper is structured as follows. In Section 2 we introduce a running example and Section 3 provides background information on process-related RBAC models. Section 4 deals with the detection of runtime consistency conflicts and Section 5 presents our conflict resolution approach. Section 6 discusses related work and Section 7 concludes.

2 Running Example

We illustrate the problem space of this paper based on a scenario from the e-health domain (see [8]). The treatment of patients is a critical task, and the patient data constitute sensitive information. In this example, we model business processes using UML activity diagrams [10], in particular the BusinessActivities extension [16]. It enables the definition of process-related RBAC models via extended UML activity models.

Figure 2 shows an exemplary patient examination process modeled as *BusinessActivity* and the corresponding process-related RBAC model. We assume that all tasks are supported by technical services. Some tasks involve human interaction. The left hand side of the figure shows the BusinessActivity model of a patient examination process, and the right hand side contains an excerpt of the RBAC definitions which apply to the scenario. We define five types of roles (i.e., *Staff, Patient, Physician, Surgeon, Administration)*, each with a list of tasks they are permitted to execute (displayed after the string "Tasks:"), and seven subjects (i.e., *Sue, Alice, Joe, Pete, John, Bob, Lisa*).

The examination process starts with the retrieval of the patient's personal data. Afterwards the patient is assigned to a physician. After the assignment the corresponding physician requests an x-ray image from the responsible department. Subsequently, the physician decides whether additional data are required (e.g., information about related injuries or diseases in the past). In that case, the historical data is requested from partner hospitals. Due to privacy reasons,

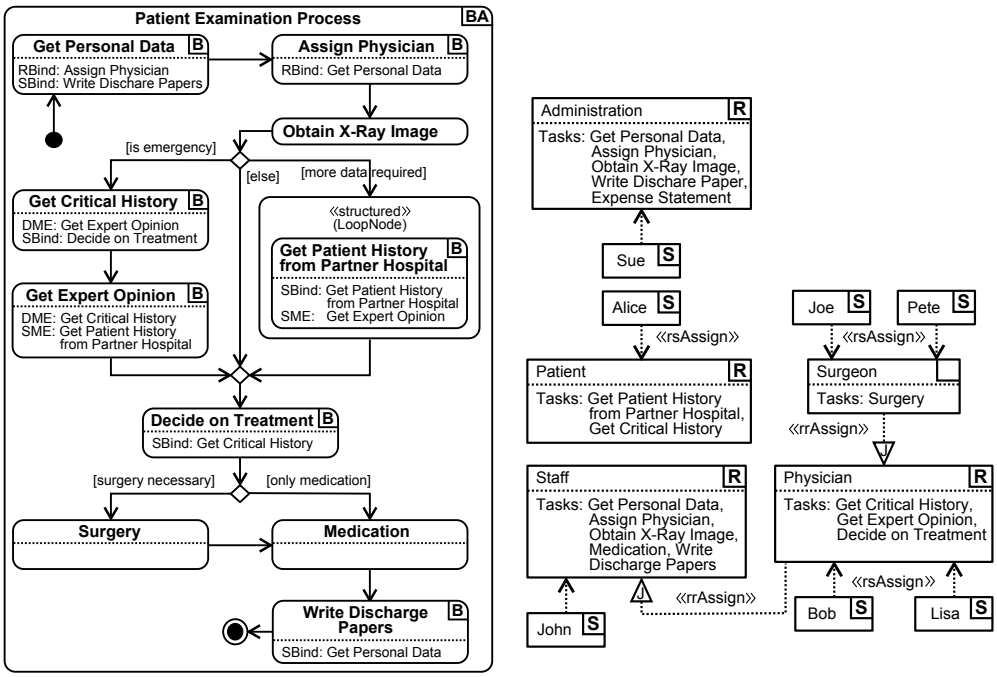


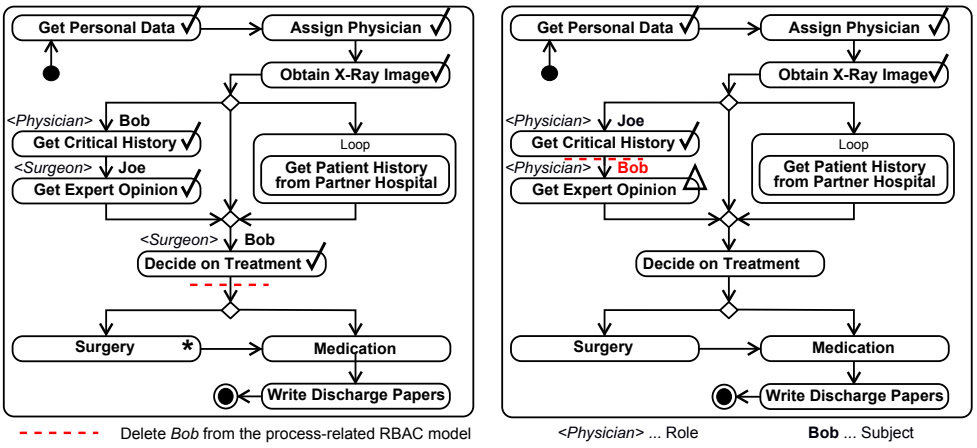
Fig. 2. Patient Examination Scenario modeled as UML Business Activity ($p_t \in P_T$)

the historical data are only disclosed to the patient. Thus, the task *Get Patient History from Partner Hospital* has to be executed by a subject with the role *Patient*. Additional data is also needed in case of an emergency. If the emergency demands for immediate treatment, it is important to determine historical data about any critical conditions or diseases that might interfere with the proposed treatment (*Get Critical History*). To avoid a single physician to make wrong decisions in an emergency it is mandatory to get the opinion of a second expert. The decision which treatment is given to the patient is finally followed by a surgery and a medication or a medication only. The process ends with the writing of the discharge papers for the patient.

In this paper, we focus on the impact of changes in process-related RBAC models on the consistency of a business process instances at runtime (on instance-level) under the assumption that the access control model is consistent [14, 15, 16] and the business process is satisfiable on the type level (see [3, 5]). At runtime the changes affect the execution of specific task instances in a way that they cannot be performed by the allocated subject without violating the process-related RBAC policy. In general, we differentiate three execution states of task instances:

– *Allocated*: Each task instance of a corresponding process instance is *allocated* before it is executed (e.g., task *Get Expert Opinion* in Figure 3(b), indicated by Δ). A task is allocated to a single subject and a specific role (see, e.g., [16]) which has to execute the task further on in the process instance.

– *Running*: A task instance which is being executed is *running* (e.g., task *Surgery* in Figure 3(a), indicated by *).
– *Finished*: A task instance which was completely executed is *finished* (e.g., task *Get Personal Data* in Figure 3(b), indicated by ✓).



(a) Process Instance $pi_v \in pi(p_t)$ (b) Process Instance $pi_w \in pi(p_t)$

Fig. 3. Exemplary Process Instances of Patient Examination Scenario (see Figure 2)

Consider that subject *Bob* is deleted from the process-related RBAC model of Figure 2. At design time this change does not violate the consistency of the RBAC model, and it does not impact the satisfiability of the corresponding process model. At runtime, this change does not take effect on process instance pi_v from Figure 3(a), because all task instances which are allocated to *Bob* are already *finished*.

In case *Get Expert Opinion* is *allocated* to *Bob* with role *Physician* at the moment *Bob* is deleted (see pi_w in Figure 3(b)), it has to be decided, if the task instance can be executed with the given allocation (see Section 4). If the task instance cannot be performed by the allocated subject this leads to a runtime consistency conflict.

3 Background: Process-related RBAC Models

The formal notations, algorithms and resolution strategies presented in the following sections are based on the formal definitions for process-related RBAC models defined in [15, 16]. In the following we provide an overview of these definitions.

Definition 1 (Process-related RBAC Model) *A process-related RBAC Model is defined as* $PRM = (E, Q, D)$*, where* $E = S \cup R \cup P_T \cup P_I \cup T_T \cup T_I, Q = rh \cup rsa \cup tra \cup es \cup er \cup ar \cup pi \cup ps \cup ti$ *and* $D = sb \cup rb \cup sme \cup dme$ *refer*

to the pairwise disjoint set of the model, to mappings that establish relationships and to binding and mutual exclusion constraints respectively.

In that context S is a set of *subjects*, R is a set of *roles*, elements of P_T and P_I are called *process type* and *process instance* respectively, T_T refers to *task types* whereas T_I is a set of *task instances*. Furthermore, the definition of subject-binding (sb), role-binding (rb), static mutual exclusion (sme) and dynamic mutual exclusion (dme) is allowed on task type level. We repeat the following definitions from [15, 16] because they facilitate the understanding of the defined algorithms in Section 4.1 and the Appendix.

1. For each task type we can create an arbitrary number of respective task instances. Thus, if we have two instances of a process, we also have two instances of the corresponding tasks. Formally: The mapping $ti : (T_T \times P_I) \mapsto \mathcal{P}(T_I)$ is called **task instantiation**. For $ti(t_T, p_I) = T_i$ we call $T_i \subseteq T_I$ set of task instances, $t_T \in T_T$ is called task type and $p_I \in P_I$ is called process instance (see [16]). The mapping $tio : T_I \mapsto T_T$ is called **task instance origin** and returns the task type a specific task instance originates from.
2. For each process type we can create an arbitrary number of corresponding process instances. Formally: The mapping $pi : P_T \mapsto \mathcal{P}(P_I)$ is called **process instantiation**. For $pi(p_T) = P_i$ we call p_T process type and $P_i \subseteq P_I$ the set of process instances instantiated from process type p_T (see [16]). The mapping $pio : P_I \mapsto P_T$ is called **process instance origin** and returns the process type $p_T \in P_T$ a process instance originates from.
3. Each process type consists of an arbitrary number of task types, and each task type can be associated with an arbitrary number of process types. Thus the process-type to task-type relation is a many-to-many relation. Formally: The mapping $ptd : P_T \mapsto \mathcal{P}(T_T)$ is called **process type definition**. For $ptd(p_T) = T_{p_T}$ we call $p_T \in P_T$ process type and $T_{p_T} \subseteq T_T$ the set of task types associated with p_T. The mapping $ptd^{-1}(t_T) = P_{t_T}$ returns a set of process types $P_{t_T} \subseteq P_T$ a specific task type $t_T \in T_T$ is part of.
4. The mapping $pid : P_I \mapsto \mathcal{P}(T_I)$ is called **process instance definition**. For $pid(p_I) = T_{p_I}$, we call $p_I \in P_I$ process instance and $T_{p_I} \subseteq T_I$ the set of task instances associated with p_I. The mapping $pid^{-1}(t_I) = p_I$ returns the process instance a specific task instance belongs to.
5. At runtime each task type is instantiated and thus allocated to one subject under a specific role which has to execute the task instance further on in the process instance. The mapping $T_I \mapsto R$ is called **executing-role**. For $er(t_i) = r$ we call $r \in R$ the executing role and $t_i \in T_I$ is called executed task instance. Analogously the mapping $T_I \mapsto S$ is called **executing-subject**. For $es(t_i) = s$ we call $s \in R$ the executing subject and $t_i \in T_I$ is called executed task instance (see [16]).
6. The mapping $rh : R \mapsto \mathcal{P}(R)$ is called **role hierarchy**. For $rh(r_s) = R_j$ we call r_s senior role and R_j the set of direct junior roles. The transitive closure rh^* defines the inheritance in the role hierarchy such that $rh^*(r_s) = R_j$ includes all direct and transitive junior roles that the senior role r_s inherits

from. For the mapping $rh^{-1}(r_j) = R_s$ we call r_j junior role and R_s the set of direct senior roles.

7. The mapping $town : R \mapsto \mathcal{P}(T_T)$ is called **task ownership**, such that for each role $r \in R$ the tasks inherited from its junior roles are included, i.e. $town(r) = \bigcup_{r_i \in rh\ *(r)} tra(r_i) \cup tra(r)$.

The following, additional definitions extend PRM (i.e., the metamodel of a process-related RBAC model) and are required to cover the novel concepts of this paper.

1. We formally define the mapping $att : S \mapsto T_T$ as **subject-related task types**. For $att(s) = T_s$ we call $T_s \subseteq T_T$ the set of task types which can be executed by a specific subject $s \in S$.
2. Formally the mapping $tip : T_I \mapsto ST$ is called **task instance progress**. For $tip(t_I) = ST$ we call $t_I \in T_I$ task instance and $ST = \{$`FINISHED`,`RUNNING`, `ALLOCATED`$\}$ the set of task instance states.
3. Formally the mapping $ati : T_T \mapsto T_I$ is called **all task instances**. For $ati(T_t) = T_i$ we call $T_i \subseteq T_I$ the set of task instance in all process instances according to the set of task types $T_t \subseteq T_T$.

4 Detecting Runtime Consistency Conflicts

We identify two different types of changes in a process-related RBAC model. *Additive changes* refer to operations that add new subjects, roles, task types, an assignment (e.g., a role-to-role assignment) or an entailment constraint (e.g., a subject-binding) to the process-related RBAC model. On the other hand, *subtractive changes* refer to operations which delete entities from the process-related RBAC model.

Table 1 provides an overview of change operations divided into *additive* and *subtractive* change operations. The second column indicates if a change operation has an impact on (a) the consistency of the corresponding process-related RBAC model and (b) the satisfiability of the corresponding business process (see, e.g., [5, 13, 14, 15]).

In this paper, we focus on the impact of change operations on the ability to finish the execution of all affected process instances. An "x" in the third column of Table 1 indicates that a change operation might potentially influence the executability of corresponding process instances. The fourth column provides a short description for each change operation and the last column refers to the generic algorithms we provide to detect possible runtime consistency conflicts caused by the respective change operations.

Each algorithm (see Section 4.1 and Appendix A) returns a list task instances that may suffer from a runtime consistency conflict. In particular, each runtime consistency conflict is represented by a pair (ti_A, ti_B), where $ti_A, ti_B \in T_I$ are task instances. task instance ti_A must not be *null* and refers to a task instance that may no longer be executed with its current executing subject as a result of

Change Operation	DT	RT	Description	A#
Additive Change Operations				
addSubject(s)			add a new subject $s \notin S$ to the RBAC model	
addRole(r)			add a new role $r \notin R$ to the RBAC model	
addTask(t)			add a new task type $t \notin T_T$ to the RBAC model	
addRSA(s,r)	x		add a new role-to-subject assignment between a subject $s \in S$ and a role $r \in R$	
addTRA(t,r)	x		add a new role-to-subject assignment between a subject $s \in S$ and a role $r \in R$	
addR2R(r_S,r_J)	x		add a new role-to-role assignment between two roles $r_S, r_J \in R$ such that $r_J \in$ rh(r_S)	
addSB(t_A,t_B)	x	x	add a new subject-binding between two task types $t_A, t_B \in T_T$	1
addRB(t_A,t_B)	x	x	add a new role-binding between two task types $t_A, t_B \in T_T$	1
addSME(t_A,t_B)	x		add a new static mutual exclusion between two task types $t_A, t_B \in T_T$	
addDME(t_A,t_B)	x	x	add a new dynamic mutual exclusion between two task types $t_A, t_B \in T_T$	1
Subtractive Change Operations				
deleteSubject(s)	x	x	delete a subject $s \in S$ from the RBAC model	2
deleteRole(r)	x	x	delete a role $r \notin R$ from the RBAC model	3
deleteTask(t)	x	x	delete a task type $t \notin T_T$ from the RBAC model	4
deleteRSA(s,r)	x	x	delete a role-to-subject assignment between a subject $s \in S$ and an existing role $r \in R$	5
deleteTRA(t,r)	x	x	delete a task-to-role assignment between a task $t \in T_T$ and a role $r \in R$	6
deleteR2R(r_S,r_J)	x	x	delete a role-to-role assignment between two roles $r_S, r_J \in R$ such that $r_J \in$ rh(r_S)	7
deleteSB(t_A,t_B)			delete a subject-binding between two task types $t_A, t_B \in T_T$	
deleteRB(t_A,t_B)			delete a role-binding between two task types $t_A, t_B \in T_T$	
deleteSME(t_A,t_B)			delete a static mutual exclusion between two task types $t_A, t_B \in T_T$	
deleteDME(t_A,t_B)			delete a dynamic mutual exclusion between two task types $t_A, t_B \in T_T$	

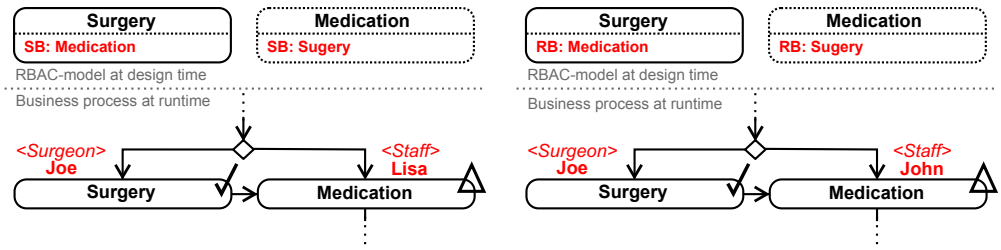
Legend: DT ... Impact at Design Time, RT ... Impact at Runtime
4.1 and the Appendix)

Table 1. Impact of Change Operations on Type- and Instance-Level

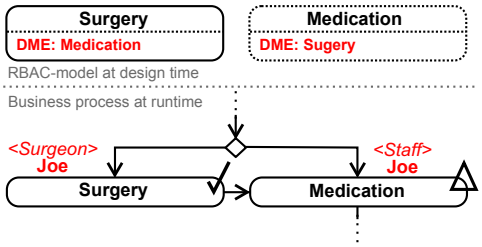
a change in the process-related RBAC model. Situations where ti_B is *not null* symbolize that the entailment constraint defined on the two task instances ti_A and ti_B cannot be correctly executed as a result of a change in the process-related RBAC model.

4.1 Detecting Runtime Consistency Conflicts in Additive Changes

We identified three additive change operations which possibly lead to runtime consistency conflicts: to *add a subject-binding* (`addSB`(t_A, t_B)), to *add a role-binding* `addRB`(t_A, t_B) and to *add a dynamic mutual exclusion* `addDME`(t_A, t_B). All other additive change operations listed in Table 1 are not considered because they do not have an impact on the runtime consistency of process instances. If we add an entity (e.g., a subject, a role, or a task type) it won't affect the executability of any process instance. If we add an assignment (e.g., a role-to-subject, a task-to-role, or a role-to-role assignment) it may have an impact on type-level but it won't affect the executability of a process instance provided that the process is satisfiable. Figure 4 shows three situations which lead to runtime consistency conflicts due to adding a subject-binding, a role-binding or a dynamic mutual exclusion constraint. Furthermore, it is not necessary to consider the adding of a static mutual exclusion constraint to cause a runtime consistency conflict because all possible conflicts are already prevented at design time per definition. If a static mutual exclusion constraint is to be defined on two task types they must not be assigned to the same role or the same subjects. It is not allowed to add a static mutual exclusion constraint on e.g. *Surgery* and *Medication* as both task types can be allocated to e.g. role *Surgeon* and thus be executed by e.g. subject *Joe* (see Figure 2).



(a) Process Instance $pi_u \in pi(p_t)$ (b) Process Instance $pi_v \in pi(p_t)$



(c) Process Instance $pi_w \in pi(p_t)$

Fig. 4. Runtime Consistency Conflicts due to the Adding of a SB, RB and DME constraint to Figure 2

Add Subject-Binding Runtime Conflict: An *Add Subject-Binding Runtime Conflict* may occur if we add a subject-binding constraint. Within a specific

process instance two task instances are already allocated to different subjects. If we want to add a subject-binding constraint on their corresponding task types we suffer an *Add Subject-Binding Runtime Conflict*.

The upper half of Figure 4(a) shows an excerpt of the process-related RBAC model from Figure 2 where a subject-binding constraint (i.e., $\texttt{addSB}(t_A, t_B)$) is to be added between the task types *Surgery* and *Medication*. At design time the corresponding business process (see Figure 2) is satisfiable as there exist several subjects (e.g., *Joe, Lisa*) that are potentially allowed to execute the subject-bound tasks.

The bottom part of Figure 4(a) shows an excerpt of a corresponding process instance where subjects *Joe* and *Lisa* have been allocated to the task instances *Surgery* and *Medication*. The subject-binding constraint between *Surgery* and *Medication* is to be added after task instance *Surgery* was executed but before task instance *Medication* is to be executed. As both task instances are supposed to be executed by the same subject, but different subjects are already allocated to the task instances, we consider this situation to be a runtime consistency conflict.

Add Role-Binding Runtime Conflict: An *Add Role-Binding Runtime Conflict* may occur if we add a role-binding constraint. Within a specific process instance two task instances are already allocated to different roles. If we want to add a role-binding constraint on their corresponding task types we suffer an *Add Role-Binding Runtime Conflict*.

In Figure 4(b) a role-binding constraint (i.e., $\texttt{addRB}(t_A, t_B)$) is to be added between the tasks *Surgery* and *Medication* on type level. In process instance pi_v *Surgery* was executed by *Joe* using the role *Surgeon* and *Medication* is already allocated to *John* and the role *Staff*. This situation leads to a runtime consistency conflict because the proposed role-binding constraint demands that the executing roles of both task instances have to be the same. Thus, executing task instance *Medication* using the currently allocated role *Staff* would violate the new role-binding constraint.

Add Dynamic Mutual Exclusion Runtime Conflict: An *Add Dynamic Mutual Exclusion Runtime Conflict* may occur if we add a dynamic mutual exclusion constraint. Within a specific process instance two task instances are already allocated to the same subject. If we want to add a dynamic mutual exclusion constraint on their corresponding task types we suffer an *Add Dynamic Mutual Exclusion Runtime Conflict*.

In Figure 4(c) a dynamic mutual exclusion constraint (i.e., $\texttt{addDME}(t_A, t_B)$) is to be added to the business process model. As this constraint requires both task instances *Surgery* and *Medication* to be performed by two *different* subjects, but the workflow engine has already allocated *Joe* to both task instances, this situation also leads to a runtime consistency conflict.

Algorithm **1** can be used to determine the set of task instances that suffer from runtime consistency conflicts as a consequence of adding an entailment constraint. It is able to detect all runtime consistency conflicts mentioned above.

Algorithm 1 Determines runtime consistency conflicts caused by $\texttt{addSB}(t_A, t_B)$, $\texttt{addRB}(t_A, t_B)$ and $\texttt{addDME}(t_A, t_B)$

Trigger: before $\texttt{addSB}(t_A, t_B)$, **before** $\texttt{addRB}(t_A, t_B)$, **before** $\texttt{addDME}(t_A, t_B)$
Input: $\{t_A, t_B\} \in T, type \in \{SB, RB, DME\}$

```
1: conflicting ← ∅
2: for all pt ∈ ptd^-1(t_A) ∩ ptd^-1(t_B) do
3:    for all pi ∈ pi(pt) do
4:       for all t_a ∈ ti(t_A, pi) do
5:          for all t_x ∈ ti(t_B, pi) do
6:             if type = SB and es(t_a) ≠ es(t_x) then
7:                conflicting ← conflicting ∪ (t_a, t_x)
8:             else if type = RB and er(t_a) ≠ er(t_x) then
9:                conflicting ← conflicting ∪ (t_a, t_x)
10:            else if type = DME and es(t_a) = es(t_x) then
11:               conflicting ← conflicting ∪ (t_a, t_x)
12:            end if
13:         end for
14:      end for
15:   end for
16: end for
17: return conflicting
```

4.2 Detecting Runtime Consistency Conflicts in Subtractive Changes

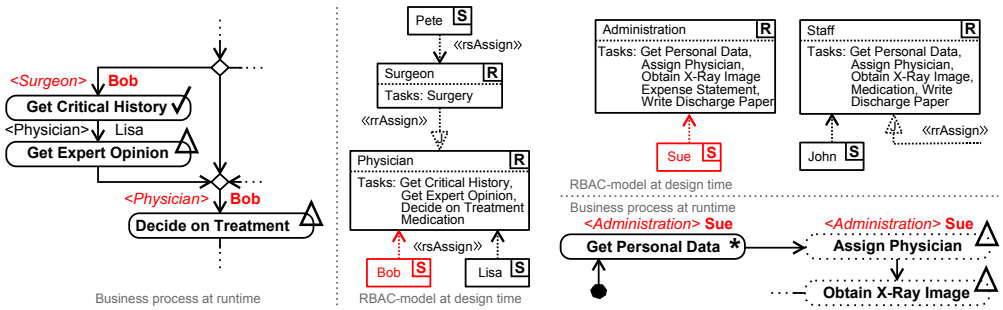
We identified six subtractive change operations that can potentially lead to runtime consistency conflicts: *delete a subject* ($\texttt{deleteSubject}(s)$), *delete a role* ($\texttt{deleteRole}(r)$), delete a task ($\texttt{deleteTask}(t)$), *delete a role-to-subject assignment* ($\texttt{delRSA}(r, s)$), *delete a task-to-role assignment* ($\texttt{delTRA}(t, r)$) and *delete a role-to-role assignment* ($\texttt{delR2R}(r_S, r_J)$). The deletion of an entailment constraint is not considered because it does not affect the execution of a process. Again, we illustrate the occurrence of runtime consistency conflicts with the help of the patient examination scenario depicted in Figure 2. The corresponding set of generic algorithms for detecting the conflicts are presented in the Appendix.

Delete Subject Runtime Conflict: A *Delete Subject Runtime Conflict* may occur if we delete a subject from a process-related RBAC model. The conflict may occur in the following three situations:

- Subject x is to be deleted from the process-related RBAC model. Each task instance whose executing subject is x leads to a *Delete Subject Runtime Conflict*.
- All subject-bound task instances whose constraining task instance was allocated to the subject to be deleted, lead to a *Delete Subject Runtime Conflict*. In particular, Figure 5(a) shows an excerpt of the process-related RBAC model where subject *Bob* is to be deleted. At design time, the corresponding business process is satisfiable: *Lisa* and *Pete* are both allowed to execute the subject-bound tasks *Get Critical History* and *Decide on Treatment*. The left hand side of Figure 5(a) shows an excerpt of process instance pi_w where *Bob* is

deleted after task instance *Get Critical History* was executed but before the subject-bound task instance *Decide on Treatment* is executed.

– Two task instance are role-bound. Both task instances are allocated to the same role. Subjects s is is the only subject assigned to that role. If we want to delete s a *Delete Subject Runtime Conflict* occurs. The upper part of Figure 5(b) shows an excerpt of the process-related RBAC model where subject *Sue* is to be deleted. Again, at design time the corresponding business process is satisfiable: any subject assigned to role *Staff* is allowed to execute the role-bound tasks *Get Personal Data* and *Assign Physician*. The bottom part of Figure 5(b) shows an excerpt of process instance pi_w where *Sue* is deleted after task instance *Get Personal Data* was executed but before the role-bound task instance *Assign Physician* is executed. As it is allocated to *Sue* with role *Administrator* this leads to a *Delete Subject Runtime Conflict*.



(a) Process Instance $pi_v \in pi(p_t)$ (b) Process Instance $pi_w \in pi(p_t)$

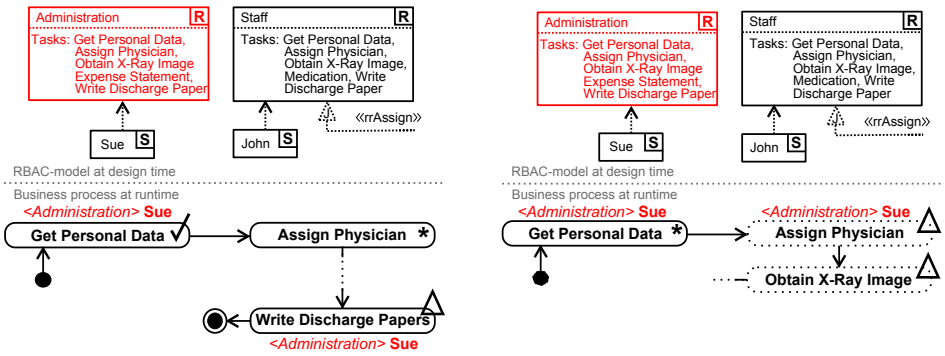
Fig. 5. Runtime Consistency Conflicts due to the Deletion of a Subject related to Figure 2

Delete Role Runtime Conflict: A *Delete Role Runtime Conflict* may occur if we delete a role from a process-related RBAC model. The conflict may occur in the following three situations:

– Role r is to be deleted from the process-related RBAC model. Each task instance whose executing role is r leads to a *Delete Role Runtime Conflict*.
– Within a specific process instance two subject-bound task instances are already allocated. The constrained task instance i is allocated to role r and any subject s assigned to r. Assume role r_D is a (transitive) junior role of r which holds the actual permissions to execute i. If we want to delete r_D we suffer an *Delete Role Runtime Conflict* if s is not allowed to execute i without the permissions of r_D.
The upper part of Figures 6(a) and 6(b) show an excerpt of the process-related RBAC model where role *Administration* is to be deleted. At design time the corresponding business process is satisfiable: any subject that owns the role *Stuff* is allowed to execute the subject-bound tasks *Get Personal Data*

and *Write Discharge Papers* and thus the entailment constraint in Figure 6(a) is enforced correctly. The bottom part of Figure 5(a) shows an excerpt of a corresponding process instance where *Administration* is deleted after task instance *Get Personal Data* was executed by *Sue* with the executing role *Administration* but before the subject-bound task instance *Write Discharge Papers* is executed.

– All role-bound task instances whose constraining task instance was allocated to the role to be deleted, leads to a *Delete Role Runtime Conflict*. In Figure 6(b) the role-binding constraint is also correctly enforced. In particular, any subject that owns the role *Stuff* is allowed to execute any of the role-bound tasks *Get Personal Data* and *Assign Physician*. The bottom part of Figure 6(b) shows an excerpt of process instance pi_v where *Administration* is deleted while task instance *Get Personal Data* and before the role-bound task instance *Assign Physician* is executed. As role *Administration* is deleted *Sue* is not allowed to perform a task which was allocated to the role we suffer a *Delete Role Runtime Conflict*



(a) Process Instance $pi_v \in pi(p_t)$ (b) Process Instance $pi_w \in pi(p_t)$

Fig. 6. Runtime Consistency Conflicts due to Removing Role *Administration* related to Figure 2

Delete Task Type Runtime Conflict: A *Delete Task Type Runtime Conflict* may occur if we delete a task type from a process-related RBAC model.

In general, each task whose corresponding task type is to be deleted leads to a *Delete Task Type Runtime Conflict* (see Figure 7). The upper part of the Figure 7 shows an excerpt of the process-related RBAC model where we want to delete task type *Write Discharge Papers*. At design time the corresponding business process is satisfiable. The bottom part of the figure shows an excerpt of a corresponding process instance where *Write Discharge Papers* is deleted while it is executed by *John* which leads to a *Delete Task Type Runtime Conflict*.

Delete Role-to-Subject Assignment Runtime Conflict: A *Delete Role-to-Subject Assignment Runtime Conflict* may occur if we delete a role-to-subject

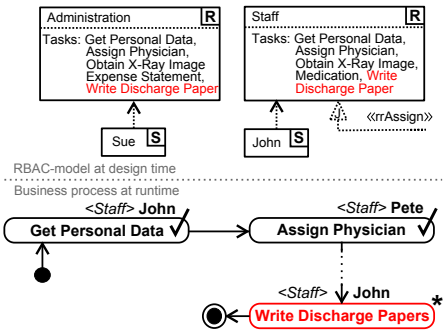


Fig. 7. Runtime Consistency Conflicts due to the Deletion of Task Type *Write Discharge Papers* related to Figure 2

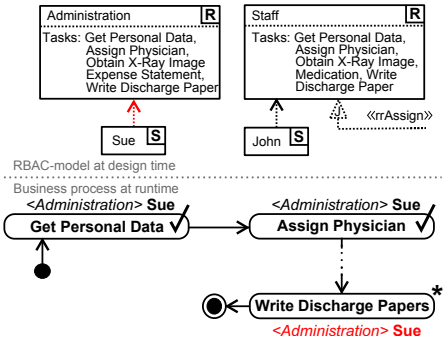


Fig. 8. Runtime Consistency Conflicts due to the Deletion of Role-to-Subject Assignment *Administration - Sue* related to Figure 2

assignment from a process-related RBAC model. The conflict may occur in the following three situations:

– Role r is assigned to subject s. Task t is assigned to r. An instance i of t is allocated to s and r or one of its (transitive) senior roles. If the role-to-subject assignment between r and s is deleted and s is allowed to perform t only via r this leads to an *Delete Role-to-Subject Assignment Runtime Conflict* when executing i. The upper part of Figure 8 shows an excerpt of the process-related RBAC model where the role-to-subject assignment of *Administration* and *Sue* is to be deleted. At design time the corresponding business process is satisfiable: any subject assigned to role *Staff* is allowed to execute *Write Discharge Papers* and thus correctly enforce RBAC policy. The bottom part of Figure 8 shows an excerpt of a corresponding process instance where the role-to-subject assignment is deleted before the task instance *Write Discharge Papers* is executed but after it was allocated to *Sue*. As *Sue* is no more assigned to *Administration* she is not allowed to execute the task instance at all.
– All subject-bound task instances whose constraining task instance suffers a *Delete Role-to-Subject Assignment Runtime Conflict* leads to the same if its allocated subject is not allowed to perform the constraining task instance due to the proposed deletion of the role-to-subject assignment. In Figure 8

there exists a subject-binding between *Get Personal Data* and *Write Discharge Papers* which cannot be correctly enforced within the specific process instance after applying the proposed deletion of the role-to-subject assignment.
– All role-bound task instances whose constraining task instance suffers a *Delete Role-to-Subject Assignment Runtime Conflict* leads to the same if its allocated role is not assigned to the constraining task instance due to the proposed deletion of the role-to-subject assignment.

Delete Task-to-Role Assignment Runtime Conflict: A *Delete Role-to-Subject Assignment Runtime Conflict* may occur if we delete a task-to-role assignment from a process-related RBAC model. The conflict is caused by the following three situations:

– Each task instance whose corresponding task type t is to be deassigned from a specific role which (or one of its transitive senior roles) at once accords to the executing role of the task instance leads to a *Delete Role-to-Subject Assignment Runtime Conflict*. The upper part of Figure 9 shows an excerpt of the process-related RBAC model where the task-to-role assignment of *Administration* and *Write Discharge Papers* is to be deleted. At design time the corresponding business process is satisfiable: any subject assigned to role *Staff* is allowed to execute *Write Discharge Papers* and thus the RBAC policy is enforced correctly. The bottom part of Figure 9 shows an excerpt of a corresponding process instance where the task-to-role assignment is deleted before the task instance *Write Discharge Papers* is executed but after it was allocated to *Sue*. As *Sue* is assigned to *Administration* she is not allowed to execute the task instance anymore.
– All subject-bound task instances whose constraining task instance suffers a *Delete Role-to-Subject Assignment Runtime Conflict* also lead to a *Delete Role-to-Subject Assignment Runtime Conflict* if the allocated subject is not allowed to perform the constraining task instance due to the deletion of the role-to-subject assignment. Figure 9 shows a subject-binding between *Get Personal Data* and *Write Discharge Papers* which cannot be correctly enforced within the specific process instance after applying the proposed deletion of the task-to-role assignment.
– All role-bound task instances whose constraining task instance suffers a *Delete Role-to-Subject Assignment Runtime Conflict* leads to the same if its allocated role is not assigned to the constraining task instance due to the proposed deletion of the task-to-role assignment.

Delete Role-to-Role Assignment Runtime Conflict: A *Delete Role-to-Subject Assignment Runtime Conflict* may occur if we delete a task-to-role assignment from a process-related RBAC model. The conflict is caused by the following three situations:

– Each task instance whose executing subject is assigned to a transitive senior role of the role whose junior role is to be deassigned leads to a conflict. The left hand side of Figure 10 shows an excerpt of the process-related RBAC model

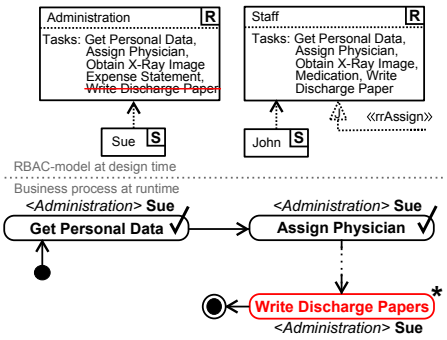


Fig. 9. Runtime Consistency Conflicts due to the Deletion of Task-to-Role Assignment *Administration - Write Discharge Papers* related to Figure 2

where the role-to-role assignment of *Surgeon* and *Physician* is to be deleted. At design time the corresponding business process is satisfiable: any subject assigned to role *Physician* is allowed to execute *Decide on Treatment* and thus the RBAC policy is enforced correctly. The right hand side of Figure 10 shows an excerpt of a corresponding process instance where the role-to-role assignment is deleted before the task instance *Decide on Treatment* is executed but after it was allocated to *Pete*. As *Pete* is not assigned to *Physician* anymore he is not allowed to execute the task.

– All subject-bound task instances whose constraining task instance suffers a *Delete Role-to-Role Assignment Runtime Conflict* leads to the same if its allocated subject is not allowed to perform the constraining task instance due to the proposed deletion of the role-to-subject assignment. In Figure 10 there exists a subject-binding between *Get Critical History* and *Decide on Treatment* which cannot be correctly enforced within the specific process instance after applying the proposed deletion of the role-to-role assignment.
– All role-bound task instances whose constraining task instance suffers a *Delete Role-to-Role Assignment Runtime Conflict* leads to the same if its allocated role is not assigned to the constraining task instance due to the proposed deletion of the task-to-role assignment.

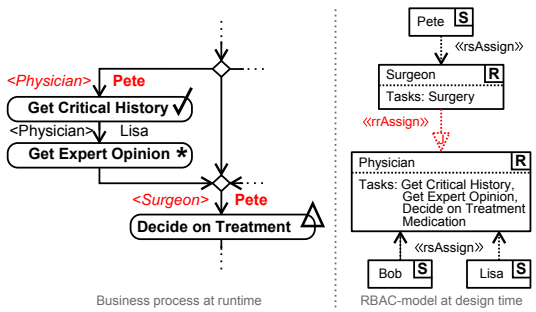


Fig. 10. Runtime Consistency Conflicts due to the Deletion of Role-to-Role Assignment *Surgeon - Physician* related to Figure 2

5 Resolving Runtime Consistency Conflicts

This section discusses possible resolution strategies for each runtime consistency conflict. Our resolution strategies rely on the following four basic *process instance management operations* which are supposed to be provided by the business process execution engine:

- `stop(`ti`)`: This procedure stops a specific task instance $ti \in T_I$ whose state is running (i.e., `tip(`ti`) = RUNNING`).
- `alloc(`ti`)`: This procedure initializes the allocation of a subject $s \in S$ and a role $r \in R$ to a specific task instance $ti \in T_I$ such that `er(`ti`)` $= r$ and `es(`ti`)` $= s$ where s is allowed to perform ti with the role r according to the corresponding process-related RBAC model. The actual selection of s and its corresponding role r is carried out by an external component (e.g., a policy decision point, see [8]) which considers the proposed change operation.
- `wait(`ti`)`: This procedure waits for a task instance $ti \in T_I$ to be finished (i.e., `tip(`ti`) = FINISHED`).
- `run(`ti`)`: This procedure runs a specific task instance $ti \in T_I$ whose state is allocated (i.e., `tip(`ti`) = ALLOCATED`).

We describe concrete resolution strategies as sequences of *process instance management operations*. For example, if we assume that subject s which is currently allocated to task instance t_a is to be deleted, a possible concrete resolution strategy could be as simple as: `alloc(`t_a`)`. That is, we use `alloc(`t_a`)` to allocate another subject to perform t_a.

The following Table 2 provides an overview of the different resolution strategies that can be applied. The resolution strategies are generic and therefore can be used for each of the runtime consistency conflicts. Instead, they depend on the current state of a conflicting task instance. Regarding the previously mentioned *Delete Subject Runtime Conflict* we have to consider the current state of the conflicting task instance t_a. If t_a is currently in the state `ALLOCATED`, Table 2 suggests the following two resolution strategies: (a) `alloc(`t_a`)` and (b) `wait(`t_a`)`.

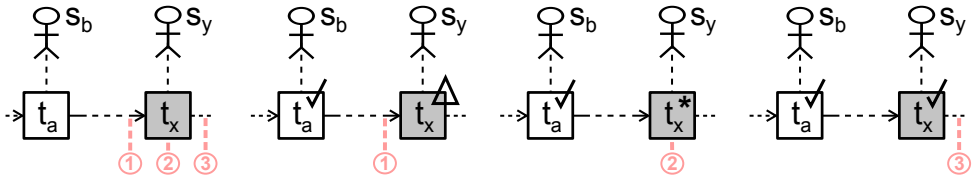
In general, from an *unconstrained* task instance point of view a change of the corresponding process-related RBAC model can be applied at one of the following task instance states (indicated by ① to ③ in Figures 11(a) to 11(d)):

- Figure 11(b) shows that the change is to be applied before t_x is executed. As the affected task instance t_x is already allocated this state is indicated as **A** (i.e., `ALLOCATED`, see Section 2).
- Figure 11(c) shows that the change is to be applied at the moment t_x is executed. As the affected task instance t_x is already running this state is indicated as **R** (i.e., `RUNNING`, see Section 2).
- Figure 11(d) shows that the change is to be applied when t_x is finished. Thus the affected task instance t_x is indicated as **F** (i.e., `FINISHED`, see Section 2).

These three states are also reflected in Figure 11. Both Figure 11 and 12 show excerpts of an exemplary business process with two task instances (i.e., t_a

Task Instance States	Resolution Strategy
Unconstrained Task Instance ($t_x = null$)	
① tip(t_a) = Allocated (A)	(a) alloc(t_a) (b) wait(t_a)
② tip(t_a) = Running (R)	(a) stop(t_a), alloc(t_a), run(t_a) (b) wait(t_a)
③ tip(t_a) = Finished (F)	-
Constrained Task Instance ($t_x \neq null$)	
④ tip(t_a) = A AND tip(t_x) = A	(a) alloc(t_a), alloc(t_x) (b) wait(t_a), wait(t_x)
⑤ tip(t_a) = R AND tip(t_x) = A	(a) stop(t_a), alloc(t_a), run(t_a), alloc(t_x) (b) wait(t_a), wait(t_x)
⑥ tip(t_a) = F AND tip(t_x) = A	(a) alloc(t_a), run(t_a)+, alloc(t_x) (b) wait(t_x)
⑦ tip(t_a) = F AND tip(t_x) = R	(a) alloc(t_a), run(t_a)+, stop(t_x), alloc(t_x), run(t_x) (b) wait(t_x)
⑧ tip(t_a) = F AND tip(t_x) = F	-

Table 2. Proposed Resolution Strategies based on Task Instance State Records (t_a, t_x) (see Figures 11 and 12)



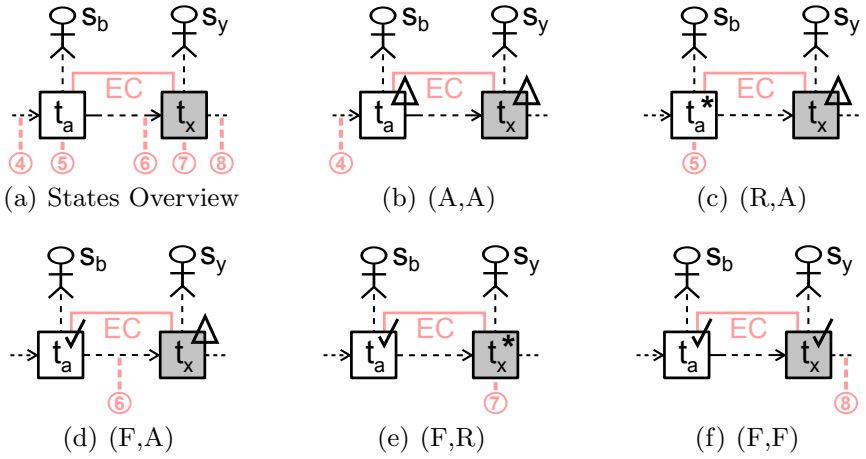
(a) States Overview (b) Allocated (A) (c) Running (R) (d) Finished (F)

Fig. 11. Changes at Different Task Execution States *without* Constraints

and t_x) and two subjects which are allocated to the tasks (via its roles). Each figure depicts the same change operation at different points in time (indicated by ① to ⑧). task t_x is shaded grey to indicate that its execution is affected by the intended change.

From a *constrained* task instance's point of view there are five different situations a change of the corresponding process-related RBAC model can be applied at runtime (indicated by ④ to ⑧ in Figures 12(b) to 12(f)). As t_a is executed before t_x it constrains the execution of t_a. Thus, t_a is called the *constraining task* and t_x is called the *constrained task*. The five different situations are as follows:

– Figure 12(b) shows that the change is to be applied before the constraining task t_a is executed. As both, t_a and t_x are allocated the situation is referred to as **(A,A)**.



(a) States Overview (b) (A,A) (c) (R,A)

(d) (F,A) (e) (F,R) (f) (F,F)

Fig. 12. Changes at Different Task Execution States *with* Constraints

- Figure 12(c) shows that the change is to be applied at the moment the constraining task t_a is executed. As t_a is running and t_x is allocated the situation is referred to as **(R,A)**.
- Figure 12(d) shows that the change is to be applied at the moment the constraining task t_a is finished but before t_x is executed. As t_x is allocated the situation is referred to as **(F,A)**.
- Figure 12(e) shows that the change is to be applied at the moment the constrained task t_a is executed. As t_a is finished and t_x is running the situation is referred to as **(F,R)**.
- Figure 12(f) shows that the change is to be applied at the moment the bots tasks t_a and t_x are finished. Thus, the situation is referred to as **(F,F)**.

The process instance management operation `run(`t_a`)+` as part of the resolution strategies for ⑥ and ⑦ implies that each task instance t_y which is (a) part of the business process flow between t_a and t_x and (b) is already executed (i.e., `tip(`t_y`) = FINISHED`) has to be executed once more as there might be dependencies between t_a and any task instance t_y related to the executing subject and the executing role of t_a (e.g., different subjects executing the same task instance might produce different output which serves as input for a subsequent task instance).

In Figure 6(a) *Get Personal Data* and *Write Discharge Papers* are role-bound. Both task instances are allocated to role *Administration*. As *Write Discharge Papers* is allocated in the moment *Administration* is to be deleted a *Delete Role Runtime Conflicts* occurs.

That situation is referred to ⑥. When we want to apply resolution strategy (a) `run(`t_a`)+` means to run all task instances which are scheduled between *Get Personal Data* and *Write Discharge Papers* (exclusive) and *finished* again. Analogously, the deletion of the Task-to-Role Assignment between role *Administration* and task *Write Discharge Papers* in Figure 9 is referred to situation

⑦. The term `run(`t_a`)+` in resolution strategy (a) means to run all task instances which are scheduled between *Get Personal Data* and *Write Discharge Papers* (inclusive) and *finished* again.

The algorithms to detect runtime consistency conflicts proposed in the paper (see Sections 4.1 and the Appendix A) all collect task instances which suffer runtime consistency conflicts.

Each proposed change operation triggers these conflict detection algorithms. Eventually, the conflict dection algorithms trigger the conflict resolution mechanism. As we can see in Table 2, there are two abstract resolution strategies. We can either try to allocate a different subject to a conflicting task instance, or we can wait for the conflicting task instance to be finished by the currently allocated subject. The final decision-making which basic resolution strategy to choose can either be fully automated or delegated to a human person (i.e., a business process responsible). This decision mainly depends on the context and type of business process. While mainly technical or short-lived processes are often suitable for a high degree of automation, long-running processes that involve many different stakeholders and sensitive environments (like a hospital) will most likely delegate the decision to a business process responsible. The reason for this is, that re-allocating and also re-running already finished task instances simply is not feasible or sensible in some situations. For instance, in our exemplary Patient Examination Scenario it would not make sense to re-run the already finished task instance *Surgery*. Instead, a business process responsible might rather decide to apply the "wait" resolution strategy in this concrete situation.

6 Related Work

Several approaches address consistency checks of entailment constraints in process-related RBAC models and the satisfiability of business processes related to a given access control model.

The ARBAC97 model proposed in [11] allows the administration of a RBAC model by providing user-role assignment, permission-role assignment and role-role assignment to define a role hierarchy. Sandhu et al. describe and outline the ARBAC967 model and the three different activities which are all required to bring users and permissions together which is best done by different administrative roles.

In [2] the authors introduce a formal model for constrained workflow systems that incorporate constraints for implementing access control policies and entailment constraints. They provide a computation of all possible dependencies between the tasks in the workflow which serves as a basis for an analysis of the satisfiability of a workflow.

Crampton and Gutin provide new methods for determining workflow satisfiability based on the concept of constraint expression in [5]. They allow to establish the complexity of solving workflow satisfiability problem (see, e.g., [18]).

Schefer et al. [14] discuss the detection of and resolution strategies for consistency conflicts related to the definition of mutual-exclusion and binding con-

straints. In [16] the authors provide an approach to model processes and process-related RBAC models which support the continuous consistency of the process-related RBAC model itself, its related entailment constraints and the corresponding process model.

The authors of [4] present a framework for describing role-based administrative models. An administrative model defines rules which control the state changes to an access control model and the data structures defined by models. Based on the concept of administrative domains they define a number of different criteria sets controlling the effect of state changes on the set of administrative domains and thus leading to different role-based administrative models.

AW-RBAC is introduced in [9] which extends the RBAC model for adaptive workflow systems. It includes change operations and a variety of objects that are subject to change within a workflow system. Administrative and operative changes can be enforced on a set of objects in a workflow system with improved security during workflow changes and reduced administration costs by the use of the AW-RBAC model.

Weber et al. [19] discuss requirements which are relevant in context of authorization in process-aware information systems (PAIS) and proposes a comprehensive access control model with a special focus on process management systems. They present an access control model which allows for the definition of access rights as needed in adaptive process management systems. In particular, the definition of user dependent and process type dependent access rights is possible

XACML has an RBAC profile to support RBAC policies. The authors of [21] extended this profile with an administrative RBAC profile referred to as XACML-ARBAC profile. It allows to enforce ARBAC policies with XACML in web service environments. Furthermore they specify concurrency requirements of an ARBAC model and introduce a concept to capture the affected roles when the permissions granted to this role are updated due to administrative operations.

Our work is complementary to the related approaches as we discuss the impacts of changes in process-related RBAC models on the consistency of the corresponding business process instances. In particular, we discuss different types of such runtime consistency conflicts, provide algorithms to detect those conflicts and introduce corresponding strategies that allow for a systematic resolution of these conflicts.

7 Conclusion

This paper has examined the impact of changes in process-related RBAC models on the respective process instances at runtime. In particular, we systematically analyzed how every possible change operation might (negatively) affect the runtime consistency of process instances. For each potentially harmful change operation we derived a generic and reusable conflict detection algorithm that detects runtime consistency conflicts and is independent of a certain software platform

or programming language. Finally, we tackled the issue of resolving runtime consistency conflicts by proposing generic resolution strategies that take the current state of conflicting task instances into account.

This paper has gone some way towards enhancing our understanding of the effects and pitfalls of changing process-related RBAC models dynamically at runtime. In addition, we want to further establish the notion of viewing process-related RBAC models as long-lived but dynamic artifacts that are constantly subject to change, instead of completely static, "deploy once and never touch it again" type of artifacts.

In summary, this paper has thrown up questions and open issues in need of further investigation. For instance, it would be interesting to study the conflict resolution's possible degree of automation. In particular, we would need a reliable instrument for automatically choosing the most sensible resolution strategy for a given runtime consistency conflict at hand. Future work might complement our work by deriving similar conflict detection algorithms for access constraints other than task-based mutual exclusion constraints and binding constraints.

References

1. F. Casati, S. Castano, and M. Fugini. Managing workflow authorization constraints through active database technology. *Information Systems Frontiers*, 3(3):319–338, Sep 2001.
2. J. Crampton. On the satisfiability of constraints in workflow systems. Technical report, Department of Mathematics, Royal Holloway, University of London, London, 2004.
3. J. Crampton. A reference monitor for workflow systems with constrained task execution. In *Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies*, SACMAT '05, pages 38–47, New York, NY, USA, 2005. ACM.
4. J. Crampton. Understanding and developing role-based administrative models. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, pages 158–167, New York, NY, USA, 2005. ACM.
5. J. Crampton and G. Gutin. Constraint expressions and workflow satisfiability. In *Proceedings of the 18th ACM Symposium on Access Control Models and Technologies*, SACMAT '13, pages 73–84, New York, NY, USA, 2013. ACM.
6. M. Dumas, W. M. van der Aalst, and A. H. ter Hofstede. *Process Aware Information Systems: Bridging People and Software Through Process Technology.* Wiley-Interscience, 2005.
7. D. Ferraiolo and R. Kuhn. Role-based access control. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.
8. W. Hummer, P. Gaubatz, M. Strembeck, U. Zdun, and S. Dustdar. Enforcement of entailment constraints in distributed service-based business processes. *Information Software Technology*, 55(11):1884–1903, Nov. 2013.
9. M. Leitner, S. Rinderle-Ma, and J. Mangler. AW-RBAC: Access control in adaptive workflow systems. In *Proceedings of the 2011 Sixth International Conference on Availability, Reliability and Security*, ARES '11, pages 27–34, Washington, DC, USA, 2011. IEEE Computer Society.
10. Object Management Group. UML 2.4.1 Superstructure, August 2011. `http://www.omg.org/spec/UML/2.4.1`.

11. R. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for role-based administration of roles. *ACM Transactions on Information System Security*, 2(1):105–135, Feb. 1999.
12. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, Feb. 1996.
13. S. Schefer, M. Strembeck, and J. Mendling. Checking satisfiability aspects of binding constraints in a business process context. In F. Daniel, K. Barkaoui, and S. Dustdar, editors, *Business Process Management Workshops*, volume 100 of *Lecture Notes in Business Information Processing*, pages 465–470. Springer Berlin Heidelberg, 2012.
14. S. Schefer, M. Strembeck, J. Mendling, and A. Baumgrass. Detecting and resolving conflicts of mutual-exclusion and binding constraints in a business process context. In *Proceedings of the 19th International Conference on Cooperative Information Systems (CoopIS)*, volume 7044 of *Lecture Notes in Computer Science (LNCS)*, pages 329–346, Berlin, Heidelberg, 2011. Springer-Verlag.
15. M. Strembeck and J. Mendling. Generic algorithms for consistency checking of mutual-exclusion and binding constraints in a business process context. In *Proceedings of the 18th International Conference on Cooperative Information Systems (CoopIS)*, volume 6426 of *Lecture Notes in Computer Science (LNCS)*, pages 204–221, Berlin, Heidelberg, 2010. Springer-Verlag.
16. M. Strembeck and J. Mendling. Modeling process-related RBAC models with extended UML activity models. *Information & Software Technology*, 53(5):456–483, 2011.
17. J. Wainer, P. Barthelmess, and A. Kumar. W-RBAC - a workflow security model incorporating controlled overriding of constraints. *International Journal of Cooperative Information Systems*, 12:2003, 2003.
18. Q. Wang and N. Li. Satisfiability and resiliency in workflow authorization systems. *ACM Trans. Inf. Syst. Secur.*, 13(4):40:1–40:35, Dec. 2010.
19. B. Weber, M. Reichert, W. Wild, S. Rinderle, B. Weber, M. Reichert, W. Wild, and S. Rinderle. Balancing flexibility and security in adaptive process management systems. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, volume 3760 of *Lecture Notes in Computer Science*, pages 59–76. Springer Berlin Heidelberg, 2005.
20. C. Wolter, A. S. C., and Meinel. Task-based entailment constraints for basic workflow patterns. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, SACMAT '08, pages 51–60, New York, NY, USA, 2008. ACM.
21. M. Xu, D. Wijesekera, and X. Zhang. Runtime administration of an RBAC profile for XACML. *IEEE Transactions on Services Computing*, 4(4):286–299, 2011.

A Appendix

In order to detect runtime consistency conflicts we need to define a supporting procedure. Procedure 1 determines the set of transitive senior roles of a specific role rx without the junior roles of r_J in case r_S is its senior role.

Algorithm 2 Determines runtime consistency conflicts caused by `deleteSubject(`s`)`

Trigger: **before** `deleteSubject(`s`)`
Input: $s \in S$

```
1: conflicting ← ∅
2: for all t_a ∈ ati(att(s)) do
3:    if es(t_a) = s then
4:       pi ← pid^{-1}(t_a)
5:       conflicting ← conflicting ∪ (t_a, null)
6:       for all tc ∈ sb(tio(t_a, pi)) do
7:          for all t_x ∈ ti(tc, pi) do
8:             conflicting ← conflicting ∪ (t_a, t_x)
9:          end for
10:      end for
11:      for all tc ∈ rb(tio(t_a, pi)) do
12:         if rown^{-1}(er(t_a)) \s = ∅ then
13:            for all t_x ∈ ti(tc, pi) do
14:               conflicting ← conflicting ∪ (t_a, t_x)
15:            end for
16:         end if
17:      end for
18:   end if
19: end for
20: return conflicting
```

Algorithm 3 Determines runtime consistency conflicts caused by `deleteRole(`r`)`

Trigger: **before** `deleteRole(`r`)`
Input: $r \in R$
1: $conflicting \leftarrow \emptyset$
2: **for all** $t_a \in$ `ati(town(`r`))` **do**
3: **if** `er(`t_a`)` $= r$ **then**
4: $pi \leftarrow$ `pid`$^{-1}$`(`t_a`)`
5: $conflicting \leftarrow conflicting \cup (t_a, null)$
6: **for all** $tc \in$ `sb(tio(`t_a, pi`))` **do**
7: **for all** $t_x \in$ `ti(`tc, pi`)` **do**
8: **if** `er(`t_x`)` $= r$ **then**
9: $conflicting \leftarrow conflicting \cup (t_a, t_x)$
10: **end if**
11: **end for**
12: **end for**
13: **for all** $tc \in$ `rb(tio(`t_a, pi`))` **do**
14: **for all** $t_x \in$ `ti(`tc, pi`)` **do**
15: $conflicting \leftarrow conflicting \cup (t_a, t_x)$
16: **end for**
17: **end for**
18: **end if**
19: **end for**
20: **return** $conflicting$

Algorithm 4 Determines runtime consistency conflicts caused by `deleteTask(`t`)`

Trigger: **before** `deleteTask(`t`)`
Input: $t \in T_T$
1: $conflicting \leftarrow \emptyset$
2: **for all** $t_a \in$ `ati(`t`)` **do**
3: $conflicting \leftarrow conflicting \cup (t_a, null)$
4: **end for**
5: **return** $conflicting$

Algorithm 5 Determines runtime consistency conflicts caused by `delTRA(`t, r`)`

Trigger: **before** `delTRA(`t, r`)`
Input: $t \in T, r \in R$
1: $conflicting \leftarrow \emptyset$
2: $allowedRoles \leftarrow \emptyset$
3: $allowedSubjects \leftarrow \emptyset$
4: **for all** $rr \in$ `tra`$^{-1}$`(`t`)` $\setminus r$ **do**
5: $allowedRoles \leftarrow allowedRoles \cup$ `sr(`rr`)`
6: **end for**
7: **for all** $rr \in allowedRoles$ **do**
8: $allowedSubjects \leftarrow allowedSubjects \cup$ `rsa`$^{-1}$`(`rr`)`
9: **end for**
10: **for all** $t_a \in$ `ati(`t`)` **do**
11: $pi \leftarrow$ `pid`$^{-1}$`(`t_a`)`
12: **if** `er(`t_a`)` $\notin allowedRoles$ **then**
13: $conflicting \leftarrow conflicting \cup (t_a, null)$
14: **if** `ec(`t`)` $\neq \emptyset$ **then**
15: **for all** $tc \in$ `sb(`t`)` **do**
16: **for all** $t_x \in$ `ti(`tc, pi`)` **do**
17: **if** `es(`t_x`)` $\notin allowedSubjects$ **then**
18: $conflicting \leftarrow conflicting \cup (t_a, t_x)$
19: **end if**
20: **end for**
21: **end for**
22: **for all** $tc \in$ `rb(`t`)` **do**
23: **for all** $t_x \in$ `ti(`tc, pi`)` **do**
24: **if** `er(`t_x`)` $\notin allowedRoles$ **then**
25: $conflicting \leftarrow conflicting \cup (t_a, t_x)$
26: **end if**
27: **end for**
28: **end for**
29: **end if**
30: **end if**
31: **end for**
32: **return** $conflicting$

Algorithm 6 Determines runtime consistency conflicts caused by `delRSA(`s, r`)`

Trigger: **before** `delRSA(`s, r`)`
Input: $s \in S, r \in R$
1: $conflicting \leftarrow \emptyset$
2: $allowedRoles \leftarrow \emptyset$
3: $affectedTasks \leftarrow \emptyset$
4: **for all** $rr \in$ `rsa(`s`)` $\setminus r$ **do**
5: $allowedRoles \leftarrow allowedRoles \cup$ `rh`*`(`rr`)`
6: **end for**
7: **for all** $rr \in$ `rh`*`(`r`)` **do**
8: $affectedTasks \leftarrow affectedTasks \cup$ `tra(`rr`)`
9: **end for**
10: **for all** $t_a \in$ `ati(`$affectedTasks$`)` **do**
11: **if** `es(`ti`)` $= s$ **and** `er(`t_a`)` $\notin$ $allowedRoles$ **then**
12: $pi \leftarrow$ `pid`$^{-1}$`(`t_a`)`
13: $conflicting \leftarrow conflicting \cup (t_a, null)$
14: **for all** $tc \in$ `sb(tio(`t_a, pi`))` **do**
15: **for all** $t_x \in$ `ti(`tc, pi`)` **do**
16: **if** `es(`t_x`)` $= s$ **and** `er(`t_x`)` $\notin$ $allowedRoles$ **then**
17: $conflicting \leftarrow conflicting \cup (t_a, t_x)$
18: **end if**
19: **end for**
20: **end for**
21: **for all** $tc \in$ `rb(tio(`t_a, pi`))` **do**
22: **for all** $t_x \in$ `ti(`tc, pi`)` **do**
23: **if** `er(`t_x`)` $\notin$ $allowedRoles$ **then**
24: $conflicting \leftarrow conflicting \cup (t_a, t_x)$
25: **end if**
26: **end for**
27: **end for**
28: **end if**
29: **end for**
30: **return** $conflicting$

Algorithm 7 Determines runtime consistency conflicts caused by `delR2R`(r_S, r_J)

Trigger: **before** `delR2R`(r_J, r_S)
Input: $\{r_J, r_S\} \in R$

1: $conflicting \leftarrow \emptyset$
2: $affectedTasks \leftarrow \emptyset$
3: **for all** $r \in$ `rh`$^*(r_J)$ **do**
4: $affectedTasks \leftarrow affectedTasks \cup$ `tra`(r)
5: **end for**
6: **for all** $t \in affectedTasks$ **do**
7: $allowedRoles \leftarrow \emptyset$
8: $allowedSubjects \leftarrow \emptyset$
9: **for all** $r \in$ `tra`$^{-1}(t)$ **do**
10: $allowedRoles \leftarrow allowedRoles \cup$ `getSeniorsWithoutR2R`(r, r_S, r_J)
11: **end for**
12: **for all** $r \in allowedRoles$ **do**
13: $allowedSubjects \leftarrow allowedSubjects \cup$ `rsa`$^{-1}(r)$
14: **end for**
15: **for all** $t_a \in$ `ati`(t) **do**
16: **if** `es`$(t_a) \notin allowedSubjects$ **or** `er`$(t_a) \notin allowedRoles$ **then**
17: $conflicting \leftarrow conflicting \cup (t_a, null)$
18: **end if**
19: **for all** $tc \in$ `sb`(t) **do**
20: **for all** $t_x \in$ `ti`(tc, pi) **do**
21: **if** `es`$(t_x) \notin allowedSubjects$ **then**
22: $conflicting \leftarrow conflicting \cup (t_a, t_x)$
23: **end if**
24: **end for**
25: **end for**
26: **for all** $tc \in$ `rb`(t) **do**
27: **for all** $t_x \in$ `ti`(tc, pi) **do**
28: **if** `er`$(t_x) \notin allowedRoles$ **then**
29: $conflicting \leftarrow conflicting \cup (t_a, t_x)$
30: **end if**
31: **end for**
32: **end for**
33: **end for**
34: **end for**
35: **return** $conflicting$

Procedure 1 Determines all roles allowed to execute a specific task considering a deletion of a role-to-role assignment

Trigger: `getSeniorsWithoutR2R`
Input: $\{rx, r_S, r_J\} \in R$

```
1: seniors ← ∅
2: if rh^-1(rx) ≠ ∅ then
3:    for all r ∈ rh^-1(rx) do
4:       if not (rx = r_J and r = r_S) then
5:          seniors ← seniors ∪ getSeniorsWithoutR2R(r, r_S, r_J)
6:       end if
7:    end for
8:    return seniors
9: else
10:    return rx
11: end if
```

Paper D

UML2 Profile and Model-Driven Approach for Supporting System Integration and Adaptation of Web Data Mashups

The subsequent paper has been published as follows:

P. Gaubatz and U. Zdun. UML2 Profile and Model-Driven Approach for Supporting System Integration and Adaptation of Web Data Mashups. In *4th International Workshop on Lightweight Integration on the Web*, Berlin, Germany, July 2012.

Using this paper we set foot in the Web and Web engineering domain. Thus, it literally paved the way for introducing our access control related approaches in the context of real-time collaborative Web applications (i.e., Paper E, Paper F, Paper G, Paper H and Paper I). In summary, we considered Web data mashups in larger system architectures, where mashups often need to be integrated with other system components, such as services, business processes, and so on. Changing parts of one of these components often require changes in many of the dependent components to be made too. Updating dependent software components manually is usually error-prone and tedious. Therefore, we proposed a model-driven development approach based on a generic UML2 profile for defining Web data mashups, including code generation techniques to remedy this problem. Our approach builds upon the notion of describing Web data mashups as microflows, i.e., very short-lived processes. Consequently, the concepts presented in Paper A, Paper B and Paper C are applicable with minor adaptations, although this has not been a key concern of this particular publication.

UML2 Profile and Model-Driven Approach for Supporting System Integration and Adaptation of Web Data Mashups

Patrick Gaubatz and Uwe Zdun

Faculty of Computer Science
University of Vienna, Vienna, Austria,
{firstname.lastname}@univie.ac.at

Abstract From a system integration perspective, Web data mashups used in larger architectures often need to be integrated with other system components, such as services, business processes, and so on. Often a change in one of these components requires changes in many of the dependent components. Similarly, an analysis of some system properties requires knowledge about other system parts than just the mashup. Such features could be implemented using the model-driven development (MDD) approach, but existing MDD approaches for mashups concentrate on modeling and execution only. To remedy this problem, we propose a generic approach based on a UML2 profile which can easily be extended to model other system parts or integrated with other existing models. It is the foundation for generating or interpreting mashup code in existing languages as well as other system parts using the MDD approach and performing system adaptation or analysis tasks based on models in a standard modeling language.

1 Introduction

Web mashups are used to combine data from different Web documents and services to create new functionality. Web data mashups concentrate on extracting and transforming data from such Web data sources and offer them as a service. Different domain-specific languages (DSLs) that are tailored specifically to facilitate the development of Web mashups (see e.g. [1–4]), model-driven approaches for Web mashups and Web data integration [5, 6], and extensions of existing behavioral modeling languages like BPEL [7, 8] have been proposed to model Web mashups.

Most approaches today concentrate on mashup modeling and execution. From a system integration perspective, they offer two means for system integration: (1) they integrate data from Web documents and services and (2) they offer their results either as Web documents or services. The larger system integration or architectural context is usually not supported any further by current approaches. For instance, Web data mashups may be used to integrate data from various internal and external information systems. Changes (e.g. of the service interface) in any of these information systems might require adaptations of the dependent Web data mashups.

The model-driven development approach (see e.g. [9]) offers a convenient way to address this problem. Via a model-driven generator, we can generate different compo-

nents from models and re-generate the code upon changes in the models. Via a model-driven interpreter we could even support model-based runtime (on-the-fly) adaptation of the mashups. Finally, the model-driven approach could be used to generate other representations of the models. For instance, we could generate a Petri Net or automata representations of the complete process and mashup behavior to analyses aspects like deadlocks or life-locks in the entire model.

From a modeling perspective, mashups are similar to areas like behavioral software modeling (see e.g. [10]) and business process modeling or workflows (see e.g. [11]). In essence, mashups can be seen as behavioral composition models similar to UML activity diagrams [12] or microflows [13] (a microflow is a short-running, non-persistent workflow [13]), with specific functionality such as extracting data from Web pages, invoking services, and combining the data retrieved from Web pages and services using scripts. Some modeling approaches that extend existing behavioral modeling languages like BPEL have been proposed [7, 8], but BPEL is designed for long-running, transactional business processes (macroflows) rather than short-running microflows.

In this paper, we propose a UML profile for mashup modeling that is based on a core package describing basic microflows as an extension of UML activity diagrams. Mashup-specific functions are added in an extension package. In this package we semi-formally modeled some of the most common mashup functionalities. The profile is designed so that it can be extended with more specific mashup functions that are provided by mashup approaches. The core contribution of this paper is a semi-formal profile for core mashup functionality as an extension of the UML2 meta-model. As a proof-of-concept we have also implemented a model-driven interpreter for the mashup profile. To explain the generalizability of our mashup modeling profile and show that it can serve as a unified modeling approach for many existing mashup approaches, we also discuss how our approach can be used in model-driven code generators to cover other existing mashup approaches.

2 Problem Description

Current Web data modeling approaches do not consider Web data mashups in a larger architectural context. For instance, the mashup may be used inside of a business process, and both mashup and process must be monitored. Figure 1 shows the architectural overview of this example scenario. In this simple architecture example, we must integrate the business process, the mashup, the used services, and the used Web sites, and provide monitoring rules for all these components as well as their deployment configurations. If we perform changes, all these artifacts might need to be changed. Keeping them consistent during development and maintenance is tedious and error-prone.

The model-driven development approach helps to overcome this problem. Unfortunately, using the model-driven approach with mashups is difficult as they are often described in proprietary modeling or script languages and there is no unified modeling approach for them that enables us to use model-driven development approaches together with mashup approaches. Standard modeling languages that provide convenient ways to model other system parts as well like the UML are usually not used (e.g. service interfaces can be modeled as extensions of class diagrams). Furthermore, the existing

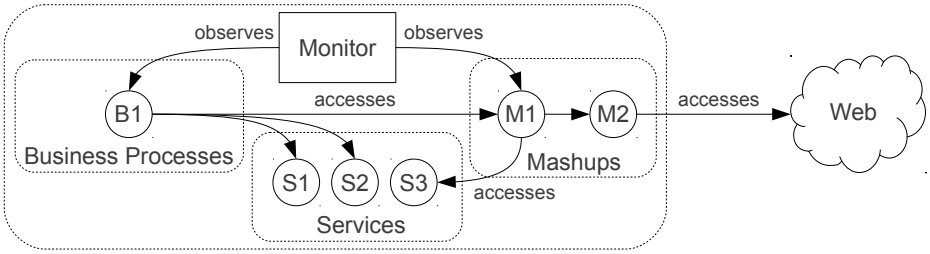


Figure 1: Architectural Overview of a System Integration Scenario

model-driven mashup approaches (e.g. [5, 6]) focus on specific aspects (like user interface layer integration) and offer only limited support for the integration of mashups in larger architectural contexts.

3 UML2 Profile for Modeling Web Data Mashups as Microflows

In order to model Web data mashups, different primitives, such as service invocations, transformation of data, and output generation in a mashup must be modeled and interconnected. To model such primitives we chose the profile extension mechanism of UML2 because there are already existing UML2 meta-classes that are semantically a close match to the characteristics of a Web data mashup. In particular, a mashup can be seen as a series of activities that perform data transformations. From the perspective of behavioral modeling, a mashup can be seen as a special purpose *microflow*: The term microflow refers to a short running, rather technical process model [13][1]. A typical way to model microflows are UML2 activity diagrams, which we will extend using a UML2 profile for modeling mashups as microflows.

This is done by semi-formally extending semantics of the respective UML2 meta-classes (rather than having to define completely new meta-classes). A profile is still valid, standard UML2. That is, it can be used in existing UML2 tools, instead of having to offer proprietary ones which are rarely used in practice. We use the Object Constraint Language (OCL) to define the necessary constraints for the defined stereotypes to precisely specify their semantics. OCL constraints are the primary mechanism for traversing UML2 models and specifying precise semantics on stereotypes.

Below, each primitive is precisely specified in the context of the UML2 meta-model using OCL constraints. This is a very important step for the practical applicability of our concepts: Without an unambiguous definition of the primitives, they cannot be used (interchangeably) in UML2 tools and model-driven generators. That is, our main reason for using the UML2 – a potential broad tool support – could otherwise not be supported.

3.1 Modeling Microflows

As a Web data mashup can be seen as a microflow, we decided to found our profile for Web data mashups on a meta-model extension for microflows. More precisely, we

[1] Microflows can be contrasted to macroflows which describe long-running, rather business-oriented process [13].

are proposing a meta-model for scripting language-based microflows in the context of service composition and service-based data integration.

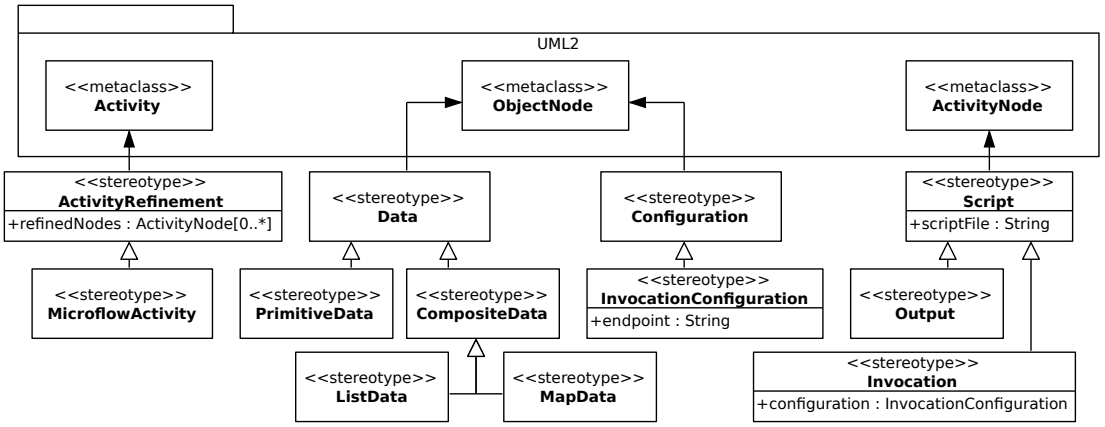


Figure 2: The Microflow Meta-Model

Figure 2 depicts the UML2 class diagram of the microflow meta-model. The *MicroflowActivity* stereotype allows us to denote an UML2 activity to be a microflow. It also allows us to make the model subject to model constraints. For example, we defined an OCL constraint (see Listing 1) specifying that an instance of a microflow must have exactly one *InitialNode* – a requirement needed to allow the execution of microflows.

```
context MicroflowActivity
  inv: self.baseActitiy.node->select(oclIsTypeOf(InitialNode))->size() = 1
context Script inv: self.scriptFile->notEmpty()
context InvocationConfiguration inv: self.endpoint->notEmpty()
context Invocation inv: self.configuration->notEmpty()
context Output inv: self.baseActivityNode.incoming->exists(in |
  Data.allInstances()->exists(data |
    in.source.oclIsTypeOf(ObjectNode) and in.source = data.baseObjectNode))
```

Listing 1: OCL Constraints for the Microflow Model

Microflows of Web data mashups read, write, transform, process, analyze, annotate, group, ... data. Consequently, our meta-model defines a *Data* stereotype. In our approach, instances of *Data* are called data objects. *Data* can either be *PrimitiveData* (e.g. strings, numbers, or boolean values) or complex *CompositeData*. The latter can either be *ListData* (i.e. arrays) or *MapData* (i.e. key/value-pairs). These two complex data structures allow us to accommodate and map (at least) the two most widely used data formats in the Web context: XML and its variations (e.g. HTML) as well as JSON.

Having introduced data objects, we have yet to define means to get them into/out of a microflow. An *Output* returns data and/or a result (e.g. an XML document) back to the executor of the microflow (e.g. a Web application). An *Invocation* is used to retrieve data to be processed from a service (e.g. a RESTful Web service).

A *Script* acts as a "placeholder" for implementation-level code. This way arbitrary extensions from existing mashup implementation languages can be integrated – allowing us to model mashups in a generalizable fashion, but still being able to incorporate the specialized features of different mashup languages via code generation. That is, the model-driven interpreter or generator will take the code in the script files and insert it at the dedicated points into the generated or interpreted code. For this reason, *Script* serves both as the meta-model's primary extension point and as a "fallback" activity. Al-

though the meta-model is extensible, in practice there will always be situations, where no "suitable" modeling-construct is available. In such cases, the developer can either extend the meta-model (i.e. introduce a new modeling-construct) or he/she directly attaches implementation-level code.

The main purpose of the *ActitivyRefinement* stereotype is to allow a *MicroflowActivity* to refine a concrete *ActivityNode*. For example, a *MicroflowActivity* (`A1`) might contain an *ActivityNode* – with the name `N1` – to be refined. A second, *MicroflowActivity* (`A2`) might then use the tag `refinedNodes` to indicate, that it refines the node `N1` (from `A1`). As we will see in Section 5, this mechanism can not only be used to refine *MicroflowActivities* but also to integrate our meta-model with other meta-models.

3.2 Modeling Web Data Mashups

Based on the rather generic microflow meta-model introduced in the previous section, we will now present a model extension aiming to cover the most basic set of invocation activities related to Web mashups (i.e. "plain" HTTP and SOAP). Note, that the resulting model is far from "complete" and mainly tries to give the reader an idea of our meta-model's extension mechanism (see Section 4 for further details).

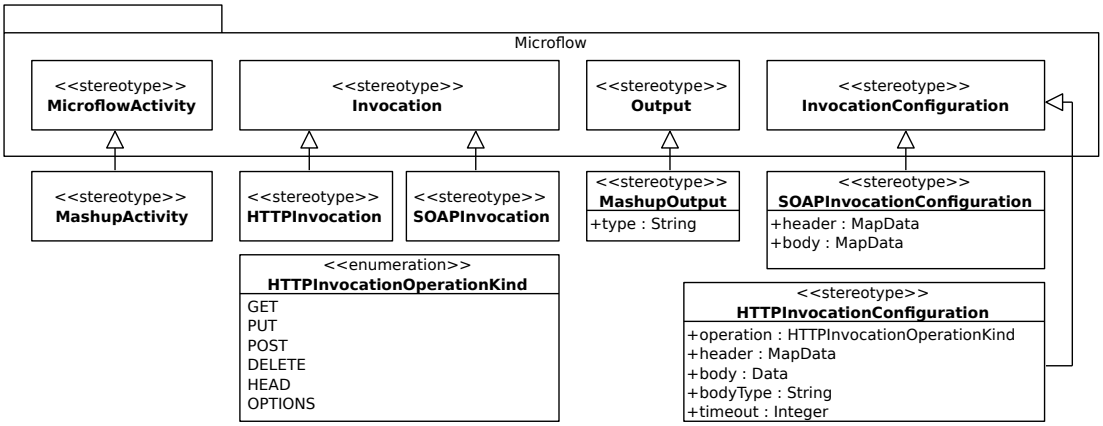


Figure 3: The Mashup Meta-Model

Figure 3 illustrates the Mashup meta-model in its UML2 class diagram representation. *Invocation* is derived twice: *HTTPInvocation* and *SOAPInvocation*. The former is used to model a plain HTTP request (e.g. to retrieve a resource from a RESTful service or to post data to a JSON-based Web service). The stereotype *SOAPInvocation* indicates an invocation of a SOAP Web service. Finally, *MashupOutput* is derived from *Output*. The mandatory `type` tag is used to specify the MIME type of the data to be returned.

```
context HTTPInvocationConfiguration
  inv: self.operation->notEmpty()
  inv: self.operation = POST or self.operation = PUT
         implies self.body->notEmpty()
  inv: self.body->notEmpty() implies self.bodyType->notEmpty()
context HTTPInvocation
  inv: self.configuration.oclIsKindOf(HTTPInvocationConfiguration)
context SOAPInvocationConfiguration inv: self.body->notEmpty()
context SOAPInvocation
  inv: self.configuration.oclIsKindOf(SOAPInvocationConfiguration)
context MashupOutput inv: self.type->notEmpty()
```

Listing 2: OCL Constraints for the Mashup Model

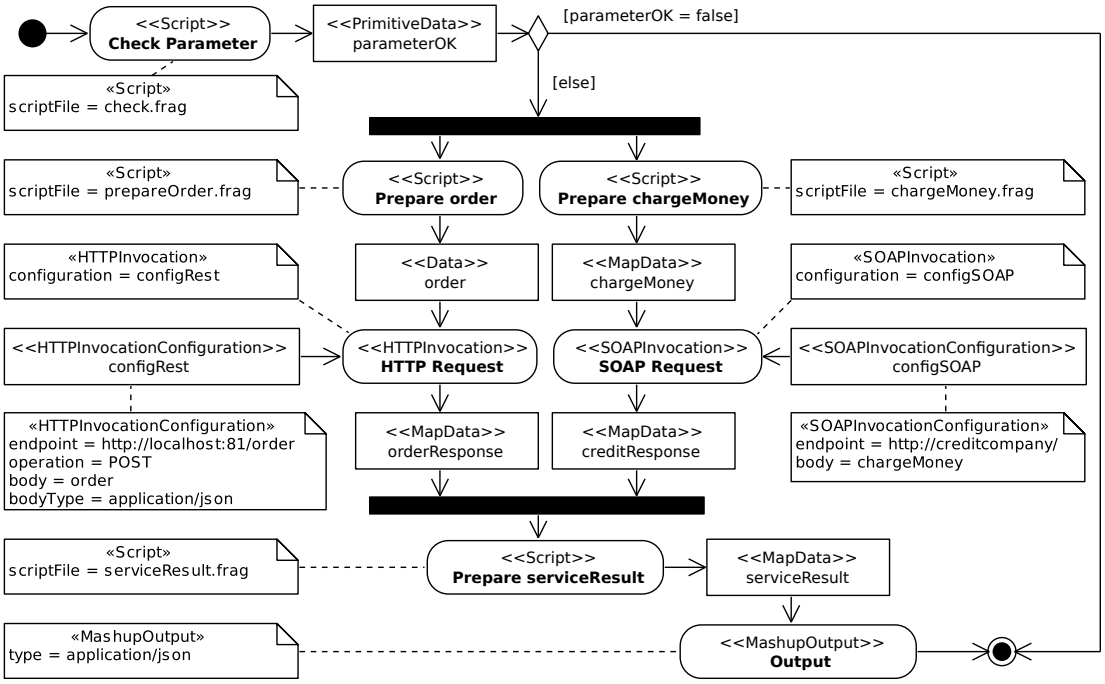


Figure 4: Example Scenario

To give you an idea how a concrete instance of our Mashup meta-model might look like, let us consider a simple online shop. An HTML page resembles its user interface. Its backend is realized using a Web data mashup. Upon invocation, the it first has to place a new order in the internal inventory system of the company, which is reachable via a JSON-based Web service. Secondly, a billing request to the external SOAP Web service of an Credit card company is issued. Finally, the result of both invocations is passed back to the user interface (e.g. a simple HTML page). Figure 4 depicts the corresponding microflow model.

4 Exploring the Generalizability of the UML2 Profile

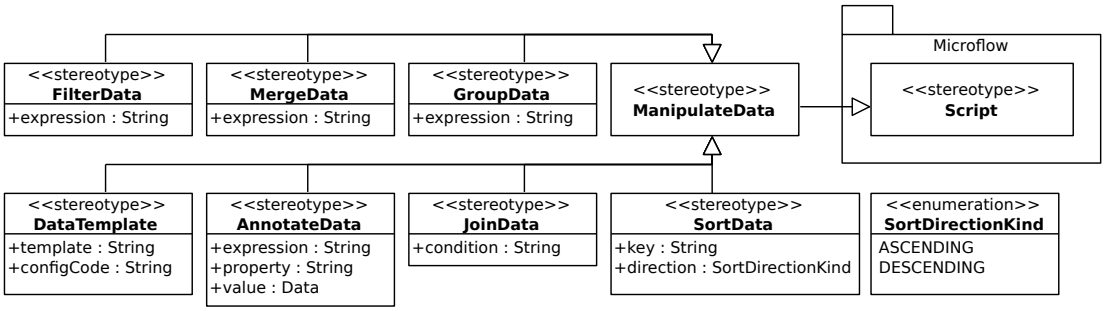


Figure 5: EMML Extensions to the Mashup Meta-Model

A generic and unified modeling approach implies, that – thanks to its generalizability – it is possible to accommodate models from similar approaches. This is achieved by mapping the model abstractions of one approach to the ones of the other. As this

is not always possible (e.g. because there is simply no matching modeling construct available), a generic modeling approach should provide an extension mechanism.

```
context ManipulateData
  inv: self.baseActivityNode.incoming->exists(in |
         Data.allInstances()->exists(data |in.source.oclIsTypeOf(ObjectNode)
           and in.source = data.baseObjectNode))
context AnnotateData
  inv: self.expression->notEmpty()
  inv: self.property->notEmpty()
  inv: self.value->notEmpty()
context FilterData inv: self.expression->notEmpty()
context GroupData inv: self.expression->notEmpty()
context JoinData
  inv: self.condition->notEmpty()
  inv: self.baseActivityNode.incoming->forAll(in |
         Data.allInstances()->select(data | in.source.oclIsTypeOf(ObjectNode)
           and in.source = data.baseObjectNode)->size() > 1
context MergeData
  inv: self.expression->notEmpty()
  inv: self.baseActivityNode.incoming->forAll(in |
         Data.allInstances()->select(data | in.source.oclIsTypeOf(ObjectNode)
           and in.source = data.baseObjectNode)->size() > 1
context SortData inv: self.key->notEmpty()
```

Listing 3: OCL Constraints for the EMML Mashup Model Extension

To explore the generalizability of our modeling approach we tried to map the concepts and model abstractions of the Enterprise Mashup Markup Language (EMML) [4]. EMML is an XML-based standard that supports the specification of processing flows for Web mashups in a platform- and vendor-independent manner. Table 1 contains a list of EMML statements (taken from the reference [4]) and shows how each statement can be mapped to our UML2 profile. In Table 1a we can see, that many statements (e.g. control flow-related) can directly be mapped to plain UML2 (e.g. `<if>`).

For a large part of the domain-specific statements (e.g. `<mashup>`) this is also the case. To cover the remaining, we had to extend our model. Figure 5 shows, that we have extended the *Script* stereotype – the primary extension point of our model – to introduce 8 new stereotypes. Listing 3 shows the corresponding OCL constraints (e.g. *JoinData* needs at least two incoming activity edges originating *Data* objects) and Table 1b shows how they are mapped to EMML. The remaining statements are listed in Table 1c. We considered them either to be "generic" in a sense that they are not very specific for the domain of "data mashups" (e.g. `<constructor>`) or to mainly exist for debugging purpose (e.g. `<assert>`). Hence, we used the *Script* "fallback" to cover them.

(a) Plain UML2

EMML	UML2
<input>	ActivityParameterNode
<variables>	ObjectNode / ObjectFlow
<include>	Activity
<macro>	
<macros>	
<if>	DecisionNode
<for>	
<foreach>	
<break>	
<while>	
<parallel>	ForkNode / JoinNode
<sequence>	ControlFlow

(b) UML2 Stereotypes

EMML	UML2
<mashup>	*MashupActivity*
<directinvoke> <invoke>	*Invocation*
<annotate>	*AnnotateData*
<filter>	*FilterData*
<group>	*GroupData*
<join>	*JoinData*
<merge>	*MergeData*
<sort>	*SortData*
<xslt>	*DataTemplate*
<output>	*Output*

(c) Script Fallbacks

EMML	UML2
<script>	*Script*
<select>	
<appendresult>	
<assert>	
<assign>	
<constructor>	
<template>	
<display>	
<datasource>	
<sql*>	

Table 1: Mapping of EMML language elements to UML2

As we could show, our modeling approach provides a model-driven abstraction that can be used to model the essence of mashups expressed in languages like EMML in a technology-independent way that supports implementing features for model-driven generation of system integration code, analysis, or adaptation based on the abstract models. EMML code could be generated from our models and Section 7 will show that it is feasible to implement a model-driven interpreter that can execute instances of our meta-model on-the-fly.

5 Integration of the UML2 Profile with Existing Models

Different meta-models can be integrated via a common meta-meta-model, like MOF for UML2. That is, every single meta-model to be integrated has to be defined using the same meta-meta-model. The profile definition mechanism of UML2 provides straightforward means to define meta-models. As a standard modeling language, lots of different UML2 profiles and UML2-derived meta-models have been proposed. Hence, basing model integration on the common UML2 meta-model allows for an straightforward integration with other UML2-based meta-models.

Using an extension of our illustrative example, we will demonstrate the model integration capabilities of our mashup meta-model via the standard UML2 extension mechanisms. As mentioned before, mashups may very likely be used in larger architectures. For instance, our example mashup from Section 3.2 may be used by a macroflow [13], a long-running, interruptible process flow which depicts the business-oriented process perspective (e.g. a business process).

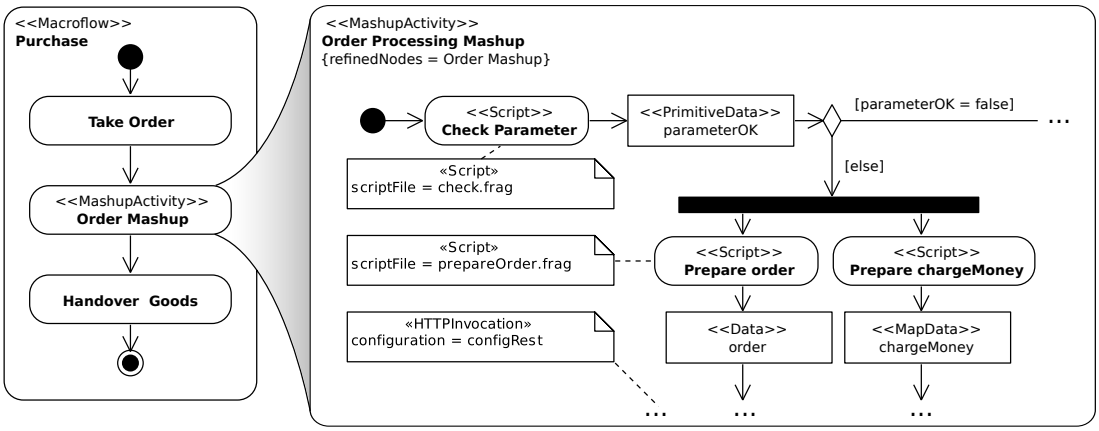


Figure 6: Integrating the Mashup Model with a Macroflow Model

Suppose that the company from our example scenario (see Section 4) also provides a physical "brick and mortar" store. The left side of Figure 6 depicts a very simplistic and high-level macroflow model of the whole buying process. The first as well as the last activities have to be conducted by a human (i.e. the shop assistant). That is, after taking the order, the original mashup model (from Section 4) shall be used to process it. Hence, we insert an activity node (*Order Mashup*) to be refined. Using the *ActivityRefinement* stereotype and the `refinedNodes` tag, we can then specify that our order processing mashup refines the mentioned activity node in the macroflow model.

This way of integrating different compatible meta-models using a tagged value introduced in the mashup profile (i.e., `refinedNodes`) is one way of model integration – in this case with other activity models. Other types of UML2 models can easily be integrated in the same way. Another option is named-based matching. For instance, the object nodes in our mashup models can easily be matched by name with the corresponding classifiers in class or component models that describe them in detail. Class models can also be used to describe the service interfaces used in a mashup.

A model-driven generator or interpreter can then use the linking tagged values or names to navigate both models and generate code for different system artifacts. The big benefit of our UML2 profile is that mashups can easily be integrated with models in other types of models and that UML2 already provides a wide variety of models that can be used to describe all kinds of other artifacts relevant for mashups.

6 Implementing a Model-driven Tool Chain

The presented UML2 meta-models have been developed and specified using a textual DSL. Frag [14, 15], a tailorable language, specifically designed for the task of defining DSLs, provides the syntactic foundation of this DSL. Among other things, Frag supports the tailoring of its object system and the extension with new language elements. Hence, it provides a good basis for defining a UML2-based textual DSL because it is easy to tailor Frag to support the definition of the UML2 meta-classes. Frag automatically provides us with a syntax for defining application models using the UML2 meta-classes. In addition Frag also provides a constraint language similar to OCL as well as a model validator. Using the model validator we can easily check a models conformance to its meta-models as well as its model constraints.

```
# define a new stereotype
FMF::Stereotype create MashupOutput \
  -superclasses Microflow::Output \
  -attributes { type String }
# define a new model constraint
MashupOutput addInvariant [notEmpty [self type]]
```

Listing 4: Frag DSL example

Note, that the textual syntax of the DSL is mainly intended to be used internally in the model validator, as a common syntax for model integration, and for debugging purposes. The developers should mainly work with UML2 and OCL tools to define the models and constraints. The main contribution of our prototypical tool chain is to validate and demonstrate that a model validation support following our concepts is feasible and can be implemented with moderate effort from scratch.

7 Implementing a Model-driven Interpreter

As a proof-of-concept, we have also implemented a basic model-driven interpreter, that is able to execute instances of our mashup meta-model on-the-fly. Using the Frag language, and mainly due to its realization of the transitive mixins concept [15], it could be implemented in roughly 450 lines of code. Mixins allow us (among other things) to add methods to classes dynamically at runtime.

```
# create executor classes
FMF::Class create MashupActivityExecutor -method execute args { ... }
FMF::Class create ScriptExecutor -method execute args { ... }
# add mixins
Mashup::MashupActivity mixins MashupActivityExecutor
Microflow::Script mixins ScriptExecutor
```

Listing 5: Defining Mixin Classes

Thus, the basic idea of our model execution-approach is to use mixins to extend our (Frag-specified) meta-model with additional execution functionality. For instance, Listing 5 shows how we define two mixin classes (`MashupActivityExecutor` and `ScriptExecutor`), both implementing the method `execute`. For every defined stereotype a corresponding executor mixin – containing the execution-logic – is needed. For instance, the execution-logic of the `MashupActivityExecutor` is to execute the model's initial node. The initial node's execution logic is to traverse its outgoing activity edge and execute the next activity node. In Listing 5 we can see, that the previously defined mixin classes are then directly attached to the classes of the meta-model (e.g. `Microflow::Script`).

```
# define a model instance
UML2::Activity create A1
Mashup::MashupActivity create M1 -baseActivity A1
# execute the model instance
M1 execute
```

Listing 6: Executing a *MashupActivity*

Having the mixin classes attached, it is then possible to directly execute any instance of our meta-model. Listing 6 depicts both the instantiation of the meta-model as well as the execution of the newly created instance via the `execute` method.

8 Related Work

A considerable amount of work has been done on the design and development of DSLs that are tailored specifically to facilitate the development of Web mashups (see e.g. [1–3]). In particular, the idea of seeing Web mashups as compositions of Web services and Web data leads to the design of numerous service composition languages. For instance, the Bite language [7] has been proposed as a simplified variant of the Web Services Business Process Execution Language (WS-BPEL) [16], a current standard technology for business process execution in the context of Web services. Like our approach this approach uses a behavioral model as the foundation of a mashup model. But BPEL is designed for long-running, transactional business processes (macroflows) and contains many language elements not useful for mashup composition, whereas our approach offers a model focused on the short-running microflows typically required for mashup composition tasks. Rosenberg et al. [8] demonstrate the applicability of Bite to model RESTful Web services and collaborative workflows.

Our model-based approach does not compete with the already existing languages and approaches. But rather it provides a model-driven abstraction that can be used to model the essence of mashups expressed in these languages. This has been demonstrated in Section 4 for the Enterprise Mashup Markup Language [4], a standard proposed by the Open Mashup Alliance. In contrast to our approach, the existing modeling

approaches are not based on a standard modeling language that provides convenient ways to model other system parts as well like the UML2 (e.g. in UML2 service interfaces can be modeled as extensions of UML2 class diagrams). Our approach can be used to augment those other mashup modeling languages with links to UML2 models for other system parts via the standard UML2 extension mechanisms.

Model-driven development in the context of Web mashups and Web data integration is nothing new and numerous approaches have been presented before. For example, Daniel et al. present mashArt [5], a model-driven approach to UI and service composition on the Web, consisting of component model for mashup components as well as an event- and flow-based service composition model. A meta-model for context-aware component-based mashup applications is presented by Pietschmann et al. [6]. The model provides means to describe all necessary application aspects on a platform-independent level, such as its components, control and data flow, layout, as well as context-aware behavior. Koch et al. present UWE [17], a model-driven approach for Web application development. The proposed UML2 profile aims to cover the entire development life cycle of Web systems and therefore clearly surpasses the scope of our own meta-model. Similarly, Kapitsaki et al. [18] also suggest a UML2 profile for modeling Web applications using UML2 class and state transition diagrams. A conceptual modeling approach to business service mashup development is presented in [19]. Bozzon et al. demonstrate the feasibility of modeling Web mashups as Business Processes using BPMN (Business Process Management Notation). In summary, these approaches attach great importance to the integration of the data and the user interface layer – which is the main focus of the meta-models of these approaches.

In contrast to these approaches, our approach tries to be as generic as possible and focus on the microflow abstraction needed to support features for model-driven generation of system integration code, analysis, or adaptation. Thus, our meta-model constitutes the bare minimum needed to model the microflows of Web mashups. Also, our main focus lies in the Web data integration and service composition aspect of Web mashups. In future extension of our model we plan to extend it to also support the user interface layer integration.

9 Conclusion and Future Work

In this paper we introduced an UML2 profile for semi-formally modeling the essence of Web data mashups based on activity diagrams and formal constraints in the OCL. We divided our meta-model into an abstract microflow layer and a mashup specific layer. We were able to show the applicability of our approach in a prototype implementation, realizing a mashup DSL and a model-driven interpreter. We showed the generalizability of our approach by mapping it to a standard mashup language, the EMML. We argued and showed how other UML2 diagrams can be integrated with our approach. Hence, the UML2 profile together with the model-driven approach help to make the mashup approach usable in a system integration context, in which the mashups and other dependent components must be changed together. The approach can potentially be used to better support the adaptation and analysis of mashups – especially together with other system components. As future work we plan to apply our approach in for these tasks.

References

1. Maximilien, E.M., Ranabahu, A., Gomadam, K.: An Online Platform for Web APIs and Service Mashups. IEEE Internet Computing **12**(5) (September 2008) 32–43
2. Vallejos, J., Huang, J., Costanza, P., De Meuter, W., D'Hondt, T.: A programming language approach for context-aware mashups. In: Proceedings of the 3rd and 4th International Workshop on Web APIs and Services Mashups. Mashups '09/'10, New York, NY, USA, ACM (2010) 4:1–4:5
3. Sabbouh, M., Higginson, J., Semy, S., Gagne, D.: Web mashup scripting language. In: Proceedings of the 16th international conference on World Wide Web. WWW '07, New York, NY, USA, ACM (2007) 1305–1306
4. Open Mashup Alliance: Enterprise Mashup Markup Language. `http://www.openmashup.org/omadocs/v1.0/`
5. Daniel, F., Casati, F., Benatallah, B., Shan, M.: Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. In: Proceedings of the 28th International Conference on Conceptual Modeling. ER '09, Berlin, Heidelberg, Springer-Verlag (2009) 428–443
6. Pietschmann, S., Tietz, V., Reimann, J., Liebing, C., Pohle, M., Meißner, K.: A metamodel for context-aware component-based mashup applications. In: Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services. iiWAS '10, New York, NY, USA, ACM (2010) 413–420
7. Curbera, F., Duftler, M., Khalaf, R., Lovell, D.: Bite: Workflow Composition for the Web. In: Proceedings of the 5th international conference on Service-Oriented Computing. ICSOC '07, Berlin, Heidelberg, Springer-Verlag (2007) 94–106
8. Rosenberg, F., Curbera, F., Duftler, M.J., Khalaf, R.: Composing RESTful Services and Collaborative Workflows: A Lightweight Approach. IEEE Internet Computing **12**(5) (September 2008) 24–31
9. Mellor, S.J., Clark, A.N., Futagami, T.: Guest Editors' Introduction: Model-Driven Development. IEEE Software **20** (2003) 14–18
10. Bock, C.: Unified Behavior Models. Journal of OO-Programming **12**(5) (1999) 65–68
11. Aguilar-Savén, R.S.: Business process modelling: Review and framework. International Journal of Production Economics **90**(2) (2004) 129 – 149
12. Object Management Group: UML 2.4.1 Superstructure. `http://www.omg.org/spec/UML/2.4.1`
13. Hentrich, C., Zdun, U.: Process-Driven SOA - Proven Patterns for Business-IT Alignment. CRC Press, Taylor and Francis, Boca Raton (2012)
14. Zdun, U.: Frag. `http://frag.sf.net/`
15. Zdun, U.: Tailorable language for behavioral composition and configuration of software components. Comput. Lang. Syst. Struct. **32**(1) (April 2006) 56–82
16. OASIS: Web Services Business Process Execution Language. `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`
17. Koch, N., Knapp, A., Zhang, G., Baumeister, H.: Uml-Based Web Engineering. In Rossi, G., Pastor, O., Schwabe, D., Olsina, L., eds.: Web Engineering: Modelling and Implementing Web Applications. Human–Computer Interaction Series. Springer London (2008) 157–191
18. Kapitsaki, G.M., Kateros, D.A., Pappas, C.A., Tselikas, N.D., Venieris, I.S.: Model-driven development of composite web applications. In: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services. iiWAS '08, New York, NY, USA, ACM (2008) 399–402
19. Bozzon, A., Brambilla, M., Facca, F.M., Carughu, G.T.: A Conceptual Modeling Approach to Business Service Mashup Development. In: Proceedings of the 2009 IEEE International Conference on Web Services. ICWS '09, Washington, DC, USA, IEEE Computer Society (2009) 751–758

Paper E

Supporting Entailment Constraints in the Context of Collaborative Web Applications

The subsequent paper has been published as follows:

P. Gaubatz and U. Zdun. Supporting Entailment Constraints in the Context of Collaborative Web Applications. In *28th Symposium On Applied Computing*, Coimbra, Portugal, March 2013.

In this paper, we presented a generic and model-driven approach for the specification and enforcement of entailment constraints in real-time collaborative Web applications. Until then, the concept of entailment constraints had only been studied in the context of workflows and business processes (including our own work presented in Paper B and Paper C). Using our CoCoForm (Constrainable Collaborative Form) prototype implementation we could demonstrate the feasibility of applying these concepts in a completely different context, i.e., real-time collaborative Web applications. An exemplary use case for CoCoForm are form-based business applications that often require forms or specific parts of forms to be filled out by different stakeholders or stakeholder roles. CoCoForm allows for precisely defining which stakeholder or stakeholder role shall be granted the permission to edit specific parts of forms. At runtime, users can fill out forms collaboratively and concurrently, while the system guarantees compliance to the defined access constraints (e.g., entailment constraints).

Supporting Entailment Constraints in the Context of Collaborative Web Applications

Patrick Gaubatz and Uwe Zdun
Faculty of Computer Science
University of Vienna, Vienna, Austria
firstname.lastname@univie.ac.at

ABSTRACT

Collaborative Web applications allow several users to collaboratively work on the same artifact. In addition to popular use cases, such as collaborative text editing, they can also be used for form-based business applications that often require forms to be filled out by different stakeholders or stakeholder roles. In this context, the different stakeholders often need to fill in different parts of the forms. For example, in an e-health application a nurse might fill in the details and a doctor needs to sign them. Role-based access control and entailment constraints provide means for defining such restrictions. So far entailment constraint have mainly been studied in the context of workflow-based architectures, but not for collaborative Web applications. We present a generic approach for the specification and enforcement of entailment constraints in collaborative Web applications that supports their real-time nature and the non-prescriptive order in which tasks can be performed. Further, we discuss a model-driven implementation approach of our concepts and lessons learned and limitations.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Web Applications; D.2.2 [**Software Engineering**]: Design Tools and Techniques

Keywords

Collaborative Web Application, Entailment Constraint, RBAC

1. INTRODUCTION

Collaborative Web applications such as Google Docs[1], Etherpad[2], or Creately[3] aim to efficiently support the joint work of different teams members, allowing them to collaboratively work on the same artifact at the same or a different time. As such collaborative Web applications are getting more and more popular, it is interesting to study their use for typical business applications that often require multiple forms to be filled out by different stakeholders or stakeholder roles. The basic approach required for such form-based collaborative applications has for instance been studied in MobWrite [8], which enables users to collaboratively fill out HTML forms. However, the – in this context crucial – aspect of access control has mainly been studied on a per-document level so far.

In form-based applications often situations occur in which different stakeholders or stakeholder roles need to fill in different parts of the forms. If restrictions on who is allowed to fill in which parts at which time exist, then proper access control must be ensured. For example, in an e-health application a nurse might fill in the details and a doctor needs to sign the document. Or a doctor files a report and a second doctor needs to check and sign the document (to realize the so-called four eyes principle). Access control in such situations has been studied in the context of role-based access control (RBAC) [9]. In RBAC, roles are used to model different job positions and scopes of duty within an information system. These roles are equipped with the permissions to perform tasks. Human users (subjects) are assigned to roles according to their work profile [11]. The examples given above have been generalized under the term entailment constraints [12], i.e. constraints that place some restriction on the subjects who can perform a task x given that a certain subject has performed another task y.

So far entailment constraints have mainly been studied in the context of workflow-based architectures (see for instance [2, 12, 14]), but not for collaborative Web applications. In this paper, we will introduce a novel approach for specifying and automatically enforcing entailment constraints in collaborative Web applications. Our approach aims to support their real-time nature and the non-prescriptive order in which tasks can be performed in collaborative Web applications through a server-side constraint checking service. On client side we propose non-intrusive modifications that integrate this constraint checking service into the collaborative Web application. A constraint model is used to integrate the client and server side in a way that requires the Web developer only to make minor modifications, whereas all security-related aspects are specified by a security expert. That is, our approach enables the separation of concerns between these two roles. In our prototype implementation we use a model-driven approach to automatically generate all required artifacts from the constraint model. That is, only the specification of the constraint model is necessary to augment a collaborative Web application with entailment constraint enforcement. Using the model-driven implementation is only an option in our approach: It is equally possible to manually hook our Web services with a few extra steps into existing collaborative Web applications.

This paper is structured as follows: In Section 2 we introduce entailment constraints. We motivate our work in Section 3. Our approach is described in Section 4. In Section 5 we discuss a pro-

[1] `https://docs.google.com`
[2] `http://etherpad.org`
[3] `http://creately.com`

SAC'13 March 18-22, 2013, Coimbra, Portugal.

totypical implementation, and Section 6 explains how it can be used to resolve the motivating example. After discussing related works in Section 7, and the lessons learned and limitations of our approach in Section 8, we conclude in Section 9.

2. ENTAILMENT CONSTRAINTS

As explained before, entailment constraints are a concept in the RBAC domain, defined as follows [12]: A *task-based entailment constraint* places some restriction on the subjects who can perform a task x given that a certain subject has performed another task y.

Different kinds of entailment constraints can be distinguished [12]: Mutual exclusion and binding constraints are typical examples of entailment constraints. They can be subdivided into *static mutual exclusion* (SME) and *dynamic mutual exclusion* (DME) constraints. An SME constraint defines that two tasks must never be assigned to the same role and must never be performed by the same subject (i.e. to prevent fraud and abuse). This constraint is global with respect to *all instances* in an information system. In contrast, DME refers to individual instances and can be enforced by defining that two tasks must never be performed by the same subject in the *same instance*. In contrast to mutual exclusion constraints, binding constraints define that two bound tasks must be performed by the *same* entity. In particular, a *subject-binding* constraint defines that the same individual who performed the first task must also perform the bound task(s). Similarly, a *role-binding* constraint defines that bound tasks must be performed by members of the same role but not necessarily by the same individual.

3. MOTIVATING EXAMPLE

In the context of health care, a medical record is used to document a patient's medical history and care. It is maintained by health care professionals (e.g. doctors) and includes information such as therapy plans, various results, and reports. There is an ongoing trend towards electronic medical records. For many of these records, a number of different health care professionals (e.g. doctors, nurses, administrative persons) have to complete a form.

Today, often the health care professionals are confronted with strict, standardized forms with precisely specified form fields, which are "hard-coded" into a custom-made (legacy) application. Adding new fields or changing existing ones is usually an error-prone and cumbersome task. An alternative solution would be workflow-based (or pageflow) applications. That is, the application consists of workflow tasks executed in a prescribed order. Each subset of the form fields, to be filled out by a specific person or user role, would be realized using a single workflow task. Executing the workflow will eventually lead to the completion of the form. The workflow-based solution has the advantage over the hard-coded solution that modifications of the control flow are possible without touching the source code. However, both solutions have a major disadvantage: Their control flows are statically prescribed at design time. It is not possible to leave this "prescribed path". This is a problem because the whole process might easily get stuck. For example, a missing signature from a doctor, who is currently off-duty, might prevent other health care professionals to proceed to the next group of form fields.

These problems led us to study in how far medical records can be created by letting users complete forms using a collaborative Web application in which ordinary HTML forms are used. In contrast to the previously described solutions a collaborative Web application would not exhibit the problems of a "prescribed path". Instead, it allows form fields to be filled out concurrently by various users at the same time. Thus, it can easily accommodate "unforeseen" deviations from the originally intended workflow.

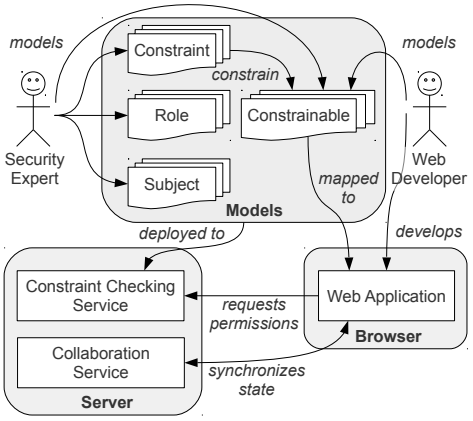


Figure 1: Architectural Overview of the Approach

This leads to the requirement to enforce *entailment constraints* in such collaborative Web applications: For some form fields it is required to precisely specify who is allowed to enter which information when. For example, a medical record form might contain an input field where the doctor in charge has to enter a documentation about the prescribed therapy. Before the final discharge of the patient, we might want the same doctor who prescribed the therapy to sign the complete medical record again. Thus, we are effectively constraining these two form fields using a *subject-binding* constraint. Another example is that for quality assurance reasons (i.e. realizing the four-eye principle) another doctor has to sign the whole record. Here, a *dynamic mutual exclusion* constraint between the two signature fields would provide means to prevent the same person to sign both fields. In this paper, we describe – to the best of our knowledge – the first approach to specify and enforce such entailment constraints in collaborative Web applications.

4. APPROACH

4.1 Approach Overview

Figure 1 gives an architectural overview of our approach. The figure illustrates two distinct stakeholder roles, Web developers and security experts, whose tasks can be clearly separated in our approach. At design time the security expert first models the roles and subjects required for the Web application (see Section 4.2). In our approach, a Web application consists of constrainable elements (e.g. a button, a JavaScript method, or even a remote service invocation). Both the Web developer and the security expert model these constrainable elements. The latter may then use constraint models to make the constrainable elements subject to various entailment constraints. The Web developer role realizes the collaborative Web application. The collaborative aspect is typically realized using a Publish-Subscriber architecture. That is, every state change of the Web application is distributed to other session participants using a collaboration service (i.e. the message broker).

From the Web developer's point of view, the development process, described so far, does not differ to the "standard approach" to developing collaborative Web applications. Only the following additional steps have to be carried out by the Web developer to guarantee compliance to the entailment constraints (see also Sec-

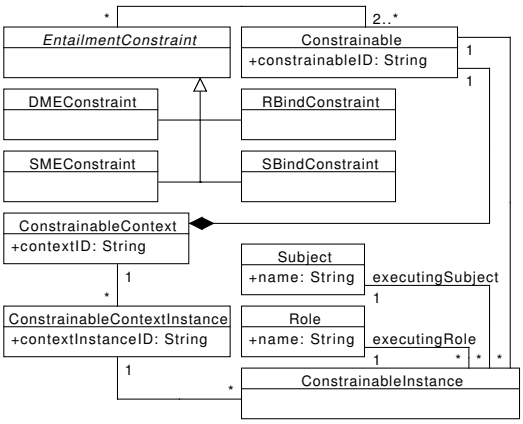


Figure 2: The Constraint Model

tion 4.3). Firstly, he or she has to map the (abstract) constrainable elements to concrete implementation-level artifacts (e.g. HTML elements) of the Web application. Whenever one of these constrained elements is invoked (e.g. a button is clicked), the Web application must request the permission to do so from a constraint checking service. This service uses a model-based constraint checking engine which either allows or denies the invocation. As the engine's decision is based on the defined subjects, roles, constraints and constrainables models, these modeling artifacts have to be deployed to the engine before. Only if the constraint service has permitted the invocation, the Web application should actually change its internal state (e.g. by calling the button's `onclick` handler) and eventually notify the session participants via the collaboration service of this particular state change. Conversely, if the invocation has been denied, the Web application must prevent the state change.

4.2 Constraint Model

The core element of our approach is a generic constraint model. Figure 2 shows an UML2 class diagram of this model. The model specifies that the *Constrainables* (mentioned before) can be constrained by an arbitrary number of *EntailmentConstraints*. Every *Constrainable* has a unique *constrainableID* and belongs to a *ConstrainableContext*. A *ConstrainableContext* has a unique *contextID*, aggregates *Constrainables*, and constitutes a self-contained execution domain or environment. More precisely, it denotes a distinct "collaboration canvas" of a Web application. For example, in a collaborative text editor there could be a *ConstrainableContext* with the ID "text document". Depending on the type of collaborative Web application, the application as a whole or just a distinct part of it constitutes a *ConstrainableContext*. We use the classes *Subject* and *Role* to model the available subjects and roles.

These model elements are usually defined at design time and instantiated at runtime using the two classes *ConstrainableContextInstances* and *ConstrainableInstance*. A *ConstrainableContext* can have an arbitrary number of *ConstrainableContextInstances*. For example, considering the previously mentioned "text document" example, a group of users may collaboratively work on a single *ConstrainableContextInstance* with a *contextInstanceID* "text document instance xyz". Whenever a *ConstrainableContextInstance* is created, a *ConstrainableInstance* has to be created for every aggregated *Constrainable* of the respective *ConstrainableContext*. That is, a *ConstrainableInstance* is an instance of exactly one *Constrainable* and is part of exactly one *ConstrainableContextInstance*.

When a *ConstrainableInstance* is invoked, the *executingRole* and *executingSubject* associations are used to establish relations to the respective *Role* and *Subject* objects actually invoking the *ConstrainableInstance*.

Method	Parameters	Return
contextInstance	contextID, contextInstanceID	–
invoke	contextID, contextInstanceID, constrainableID, subject, role	Boolean

Table 1: Interface Excerpt of the Constraint Checking Service

4.3 Runtime Enforcement and Architecture

In the previous section we have presented a generic model for defining abstract constrainable elements and making them subject to different types of entailment constraints. This section discusses the runtime architecture needed to actually enforce the compliance of the Web application with regard to the defined constraints. In essence, our approach requires both, a dedicated server-side component and modifications to the client-side application logic.

4.3.1 Server-side Constraint Checking Service

On server side, a generic and self-contained service realizes the Policy Decision Point (PDP) [7], the entity that actually decides if an invocation is to be allowed or not according to defined entailment constraints. Generally, a PDP needs to know about existing subjects, roles, and policies to be able to actually make decisions. Hence, the service needs to have access to the full set of modeling artifacts created by the security expert at design time.

Table 1 lists two essential methods that the constraint checking service has to provide to the Web application. Firstly, the Web application must call the `contextInstance` method, before any constraint checking can be done at all. It also has to provide both, a `contextID` (e.g. "text document") of an existing *ConstrainableContext* and a unique `contextInstanceID` (e.g. "text document xyz"). The service is then able to instantiate and initialize the classes *ConstrainableContextInstance* and *ConstrainableInstance*.

After `contextInstance` has been called, the service is initialized. The second mandatory method, `invoke`, must be called (by the Web application) before a constrainable element is invoked. Note that this method requires numerous parameters to be supplied: The service needs to know which subject (`subject`), using which role (`role`) is going to invoke a specific constrainable element (`constrainableID`). Furthermore, a context instance (`contextInstanceID`) and a context (`contextID`) need to be specified. If the invocation is allowed (i.e. no entailment constraints are violated), the service will then respond with the Boolean value `true`, and the Web application may finally perform the actual invocation of the constrainable element. Otherwise `false` is returned, and the invocation must be prevented. Whenever an invocation is allowed, the service will assign the *executingSubject* and *executingRole* relations of the corresponding *ConstrainableInstance* object, according to the provided `subject` and `role` parameters which is required by the underlying constraint checking algorithms (see [12] for details on the algorithms).

4.3.2 Client-side Modifications

In addition to the server-side constraint checking service, our approach also requires making modifications of the client-side application logic. These modifications would typically be performed

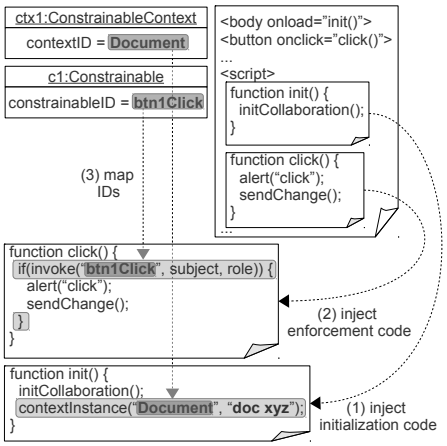


Figure 3: Required Mappings and Modifications

automatically using a model-driven code generator (see Section 5 for a discussion of our prototype). However, it is also possible to perform the modifications manually. This way it is possible to hook our Web services into existing collaborative Web applications using the few additional steps described in this section.

Figure 3 illustrates the required modifications. In general, the Web developer has to (1) call the `contextInstance` method and (2) use the `invoke` method of the constraint checking service and enforce its decision whenever a constrainable element is invoked.

Figure 3 illustrates a collaborative document editor application, in which our constraint model contains a *ConstrainableContext* with a *contextID* "Document" and an (examplary) constrainable element with a *constrainableID* "btn1Click". The figure shows an excerpt of the application's main HTML file. We can see that the browser will execute the `init()` method as soon as the `<body>` element has been parsed. The embodied `initCollaboration()` method initializes the Web application's collaboration functionality. Furthermore, there is a `<button>`. When it is clicked, an alert box is shown and this state change is propagated to other session participants using `sendChange()`.

The first required modification is the injection of the initialization code in which we have to call the `contextInstance` method of the constraint checking service. In the example, we specify that the constaint service should create an instance of the "Document" *ConstrainableContext* from our defined model and give the instance an ID of "doc xyz". Next, we have to inject the actual enforcement code, which is done by inserting a call to the constraint service's `invoke` method into the `click()` method. The result of this modification is that the original method body will only be executed, if the constraint service allows it.

5. THE COCOFORM IMPLEMENTATION

This section discusses a concrete implementation of the previously described approach. The developed prototype is called Constrainable Collaborative Forms (CoCoForm)[4] and can be used to realize the e-health record case from Section 3. The basic idea is that an ordinary HTML form (e.g. an electronic health record form)

[4]A (proof-of-concept) CoCoForm demo application is available at `http://demo.swa.univie.ac.at/cocoform`

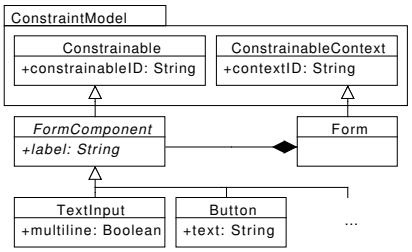


Figure 4: The WebForm Model

can be filled out collaboratively by different users at the same time. Parts of this form are subject to entailment constraints.

We decided to use a model-driven development approach for the implementation. Hence, we extended the Constraint model (see Figure 2), as can be seen in Figure 4. Every instance of *Form* constitutes a self-contained *ConstrainableContext*. A *Form* aggregates *FormComponents*. More precisely, a *FormComponent* can be a *Button*, a *TextInput*, and so on. As these components are subtypes of *Constrainable*, they can be constrained by entailment constraints.

Both models, the Constraint and the WebForm model, have been implemented in Frag [15], a Java-based, interpreted, tailorable language, specifically designed for the task of model-driven development. Frag supports both model-driven generation and interpretation of models at runtime. Hence, we have also implemented the model-based, runtime constraint checking engine in Frag. The next step was the development of the constraint checking service. We chose a RESTful service interface design, implemented in Java, using the JAX-RS API[5] and Jetty[6] as our servlet container. The service returns JSON data and is merely a HTTP-based connector between the Web application and the constraint checking engine.

The actual Web application consists of a single HTML5 document and a generic JavaScript library. We use the Open Cooperative Web Framework [13] for all collaborative aspects of our Web application. It consists of a JavaScript library, a Java servlet (i.e. the *Collaboration Service* component depicted in Figure 1), and realizes a Publish-Subscriber architecture.

For the mapping of the constrainable elements to concrete implementation-level artifact (see Section 4.3.2), the CoCoForm implementation leverages a model-driven code generator. It is used to automatically generate an instantly deployable HTML5 skeleton document from a WebForm model instance. The actual mapping information of each *FormComponent* (i.e. the constrainable element) and each *Form* (i.e. the constrainable context) is attached to the corresponding HTML5 tags. More specifically, we annotate the tags using custom (HTML5) `data-*` attributes. For example, an instance of *Form* with a *contextID* "f1" will result in a `<form data-context-id="f1">` tag. Analogously, an instance of *Button* with a *constrainableID* "b1" will be transformed to `<button data-constrainable-id="b1">`.

At runtime, the generic JavaScript library then uses these attributes to automatically register `onclick` (for buttons) and `onchange` (for text input fields) handlers for the corresponding elements. Whenever these callback functions are executed (e.g. a button has been clicked), the application calls the constraint

[5]JAX-RS, `http://jax-rs-spec.java.net`
[6]Jetty, `http://eclipse.org/jetty`

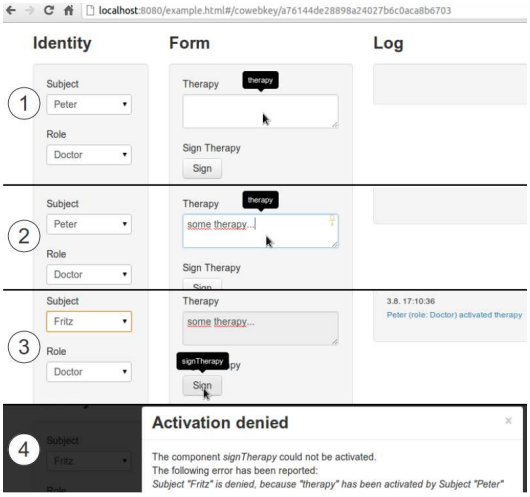


Figure 5: Subject-Binding with CoCoForm

service's `invoke` method and enforces the returned decision. If the invocation is allowed, the application disables the corresponding *FormComponent* to prevent further editing (i.e. components can only be invoked once). Secondly, the state of the component (e.g. the actual `value` of the input field, as well as the `disabled` flag) is distributed to the other session participants.

6. MOTIVATING EXAMPLE RESOLVED

Let us revisit the motivating example from Section 3. We will discuss the CoCoForm implementation of the subject-binding example from Section 3. Figure 5 shows a few screenshot excerpts of an example form. There are two form components: a text input field, which is used to document a patient's therapy, and a button, which is used to sign the documentation. For these two components a subject-binding constraint has been defined. In the first screenshot we can see the empty form. Next, *Peter*, a *Doctor*, fills out the therapy text input field. At the same time, *Fritz* joins the session and sees that the input field has already been filled out by *Peter*. Additionally, he tries to sign this form. However, he receives an error message, saying that he is not allowed to sign. This is due to the subject-binding constraint, which eventually requires *Peter* to sign. In a similar way, CoCoForm supports the definition of all entailment constraints required in the e-health case.

7. RELATED WORK

There are already some frameworks and libraries that facilitate the development of collaborative Web applications. For instance, the Open Cooperative Web Framework [13] consists of a set of JavaScript libraries and a generic Java servlet. The beWeeVee SDK [3] is a .NET-based framework and requires the Microsoft Silverlight browser plugin to be installed. MobWrite [8] is another approach for enabling real-time collaboration. However, it is restricted to synchronizing HTML forms, and the reusability and applicability is thus somewhat limited. Heinricht et al. [4] present a generic collaboration infrastructure aimed at transforming existing single-user Web applications into collaborative multi-user Web applications. In principle, our approach embraces the usage of already existing libraries and approaches. In fact, we used the Open Cooperative Web Framework to implement the collaboration aspects of CoCoForm (see Section 5). However, there is one requirement: The synchronization process of the library/framework must be interceptable. More precisely, we must prevent state changes, which have not been permitted by the constraint checking service, to be synchronized. Thus, these service invocations have to be conducted before any synchronization takes place.

The concept of task-based entailment constraints originally originates the domain of business processes and workflows. Bertino et al. [2] introduce the notion of assigning roles or subjects to tasks in a workflow and making them subject to *separation of duty* constraints. Wainer et al. [14] propose a system architecture that clearly separates the permission service from the workflow engine. Furthermore, they also present a modeling solution for specifying *binding of duty* constraints. Strembeck et al. [12] present a set of generic algorithms that ensure the consistency of entailment constraints. We used these algorithms to implement our constraint checking engine (see Section 5). In general, the existing literature – almost exclusively – examines entailment constraints in a workflow and business process context. As a result, the presented solutions are aligned with concepts that are specific for these contexts. However, in the context of collaborative Web applications, we can not resort to concepts like *task*, *process instance*, and so on. Hence, we generalized the already existing works and proposed a generic model (see Section 4.2). Instead of constraining tasks in a process, in our approach we are constraining abstract constrainable elements, which have to be mapped to concrete implementation-level artifacts.

A lot of work has been conducted in the area of RBAC in the context of Web applications and services. Ahn et al. [1] present an approach for injecting RBAC into an already existing Web-based workflow system. They propose a special reverse proxy that is able to enforce RBAC rules transparently to the actual Web application behind. Sohr et al. [10] and Hummer et al. [5] propose a similar approach in the context of Web Services. More precisely, they present generic interceptors that can be plugged into (Java-based) Web service stacks. These interceptors intercept service invocations and are then able to prevent the actual invocation in case of a policy violation. Again, this happens transparently to the underlying service implementation. In contrast, our approach requires modifications of the original client-side application logic to be made. This drawback is due to the way modern HTML5/JavaScript-based Web applications work. However, the usage of Aspect-oriented programming could help mitigating this problem (see Section 8).

8. LESSONS LEARNED

Using the CoCoForm prototype (see Section 5) we have been able to demonstrate the feasibility of our approach (see Section 1). More precisely, we showed that the concept of task-based entailment constraints can be adapted to fit into the context of collaborative Web applications. In the following paragraphs we want to discuss our lessons learned and the limitations of our approach.

The genericity of the proposed constraint model (see Section 4.2) and the constraint checking service (see Section 4.3) allow our approach to be applied to many different types of collaborative Web applications. Moreover, a single instance of the constraint checking service can potentially handle an arbitrary number of Web application instances. Thus, it is sufficient for an organization to maintain one instance of the service (maybe in replicated form).

Another positive aspect of our approach is that it follows the principle of separation of concerns. That is, the definition of roles, subjects, and constraints is completely decoupled from the actual

application. Thus, a security expert does not need to care about any implementation-level artifacts at all, whereas the Web developer does not need to care about anything related to RBAC.

We have implemented our approach with model-driven techniques to automate the generation of all additional constraint checking code. It is also possible to use the approach using manual modifications of the Web application. That is, existing code can also be instrumented this way and used with our approach by manually following the steps illustrated in Figure 3.

There are also some limitations. Firstly, our approach induces a slight performance penalty due to the required extra call of the constraint checking service. However, in the context of collaborative Web applications, which are innately prone to requiring lots of service calls, the effect of this extra call is more or less negligible.

Another limitation are the needed adaptations of the client-side application (see Section 4.3.2). The code, needed to enforce the constraint checking service's decision, has to be embedded directly into the application logic. This results in scattered and tangled code which is hard to maintain. As we have already pointed out, a model-driven code generator can avoid this issue. Besides that, aspect-oriented programming (see, e.g. [6]) can be used to decouple the enforcement-related code from the actual application code.

In our approach, the client application handles enforcement. From a security perspective, however, we often cannot trust code that is executed on the client (i.e. in a Web browser). The reason is that we cannot prevent a potential attacker from modifying the code to be executed. For instance, let us assume that there is a JavaScript method that calls the constraint checking service. An attacker might effectively undermine the enforcement just by overwriting this method and preventing the service from getting called. The effects of such client-side code injections can be contained by preventing any unauthorized state change from being distributed to other session participants. This can be achieved by using public-key cryptography. That is, the constraint checking service returns digitally signed permission documents in which the signature covers both, the actual decision and all parameters. Whenever the Web application wants to send a synchronization event to the collaboration service, it has to attach the signed permission to the request. The latter is then routed through an enforcement proxy. This proxy will only forward the request to the service, if the signature is valid (i.e. the permission document has not been tampered with) and the invocation has been permitted by the constraint checking service. In general, all (server-side) services belonging to the Web application must be tunnelled through the enforcement proxy. With these modifications we can guarantee that client-side code injections do not lead to a server-side state change or an impact on session participants.

9. CONCLUSIONS

Our approach demonstrates that the concept of task-based entailment constraints can be adapted to fit into the context of collaborative Web applications. That is, we can support collaborative editing of form-based applications with no prescribed order, and precisely specify constraints on who can perform which tasks when. We presented a generic approach that can be applied to many different collaborative Web applications. It requires some modifications to existing Web applications, but these – as well as the generation of all other required artifacts – can optionally be automated with model-driven development techniques. Our approach introduces some security concerns in untrusted environments, but these can be mitigated using public-key cryptography (see Section 8).

As future work, we will address these limitations and try to explore and establish the concept of RBAC in the context of collaborative Web application further. Furthermore, we will apply our approach to other types of collaborative processes. In particular with regard to dynamic processes (e.g. text editing or modeling) we will have to deal with completely dynamic document and constraint models (i.e. models that change at runtime).

10. REFERENCES

[1] G.-J. Ahn, R. Sandhu, M. Kang, and J. Park. Injecting rbac to secure a web-based workflow system. In *Proceedings of the fifth ACM workshop on Role-based access control*, RBAC '00, pages 1–10, New York, NY, USA, 2000. ACM.

[2] E. Bertino, E. Ferraria, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, 2(1):65–104, 1999.

[3] BeWeeVee. BeWeeVee – Life collaboration framework. `http://www.beweevee.com/`.

[4] M. Heinrich, F. Lehmann, T. Springer, and M. Gaedke. Exploiting single-user web applications for shared editing: a generic transformation approach. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 1057–1066, New York, NY, USA, 2012. ACM.

[5] W. Hummer, P. Gaubatz, M. Strembeck, U. Zdun, and S. Dustdar. An integrated approach for identity and access management in a soa context. In *16th ACM Symposium on Access Control Models and Technologies*, 2011.

[6] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In M. Aksit and S. Matsuoka, editors, *ECOOP'97 Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer Berlin / Heidelberg, 1997. 10.1007/BFb0053381.

[7] S. Kunz, S. Evdokimov, B. Fabian, B. Stieger, and M. Strembeck. Role-based access control for information federations in the industrial service sector. In *ECIS*, 2010.

[8] MobWrite. MobWrite - Real-time Synchronization and Collaboration Service. `http://code.google.com/p/google-mobwrite/`.

[9] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *Computer*, 29(2):38 –47, 1996.

[10] K. Sohr, T. Mustafa, X. Bao, and G.-J. Ahn. Enforcing role-based access control policies in web services with uml and ocl. In *Proceedings of the 2008 Annual Computer Security Applications Conference*, ACSAC '08, pages 257–266, Washington, DC, 2008. IEEE Computer Society.

[11] M. Strembeck. Scenario-driven Role Engineering. *IEEE Security & Privacy*, 8(1), January/February 2010.

[12] M. Strembeck and J. Mendling. Generic algorithms for consistency checking of mutual-exclusion and binding constraints in a business process context. In *Proceedings of the 2010 international conference on On the move to meaningful internet systems - Volume Part I*, OTM'10, pages 204–221, Berlin, Heidelberg, 2010. Springer-Verlag.

[13] The Dojo Foundation. Open Cooperative Web Framework. `http://opencoweb.org/`.

[14] J. Wainer, P. Barthelmes, and A. Kumar. W-RBAC - A Workflow Security Model Incorporating Controlled Overriding of Constraints. *International Journal of Cooperative Information Systems (IJCIS)*, 12(4), Dec 2003.

[15] U. Zdun. Frag. `http://frag.sf.net/`.

Paper F

Supporting Customized Views for Enforcing Access Control Constraints in Real-time Collaborative Web Applications

The subsequent paper has been published as follows:

P. Gaubatz, W. Hummer, U. Zdun, and M. Strembeck. Supporting Customized Views for Enforcing Access Control Constraints in Real-time Collaborative Web Applications. In *13th International Conference on Web Engineering*, Aalborg, Denmark, July 2013.

This paper is an extension of the work presented in Paper E. While Paper E laid the foundations for supporting access constraints (such as entailment constraints) in real-time collaborative Web applications, in Paper F we focused on the enforcement aspect in such environments. More precisely, we introduced the notion of leveraging the concept of dynamic view customization as a means for enforcing access constraints in real-time collaborative Web applications. We showed how this view customization approach may be implemented using CoCoForm in order to customize the views of each user in such a way that they are not able to violate any defined access constraints at all. Finally, we also provided evidence (using our CoCoForm prototype implementation) that our approach is potentially able to handle large numbers of users simultaneously connected clients with acceptable performance overhead.

Supporting Customized Views for Enforcing Access Control Constraints in Real-time Collaborative Web Applications

Patrick Gaubatz[1], Waldemar Hummer[2], Uwe Zdun[1] and Mark Strembeck[3]

[1]Faculty of Computer Science, University of Vienna, Austria
{firstname.lastname}@univie.ac.at
[2]Distributed Systems Group, Vienna University of Technology, Austria
{lastname}@infosys.tuwien.ac.at
[3]Institute for Information Systems, WU Vienna, Austria
{firstname.lastname}@wu.ac.at

Abstract Real-time collaborative Web applications allow multiple users to concurrently work on a shared document. In addition to popular use cases, such as collaborative text editing, they can also be used for form-based business applications that often require forms to be filled out by different stakeholders. In this context, different users typically need to fill in different parts of a form. Role-based access control and entailment constraints provide means for defining such restrictions. Major challenges in the context of integrating collaborative Web applications with access control restrictions are how to support changes of the configuration of access constrained UI elements at runtime, realizing acceptable performance and update behaviour, and an easy integration with existing Web applications. In this paper, we address these challenges through a novel approach supporting constrained and customized UI views that support runtime changes and integrate well with existing Web applications. Using a prototypical implementation, we show that the approach provides acceptable update behaviour and requires only a small performance overhead for the access control tasks with linear scalability.

1 Introduction

Real-time collaborative Web applications such as Google Docs[1], Etherpad[2], or Creately[3] aim to efficiently support the joint work of different team members, allowing them to collaboratively work on the same artifact at the same time. In addition to such popular examples, the real-time collaboration approach can also be used in typical business applications that often require multiple forms to be filled out by different stakeholders [7]. A crucial – though in the context of real-time collaborative Web applications often neglected – aspect of these business applications is access control.

[1] https://docs.google.com
[2] http://etherpad.org
[3] http://creately.com

In recent years, role-based access control (RBAC) [14] emerged as a standard for access control in software systems. In RBAC, roles are used to model different job positions and scopes of duty within an information system. These roles are equipped with permissions to perform tasks. Human users (subjects) are assigned to roles according to their work profile [17]. For example, in an e-health application only a doctor shall be allowed to file a report. Moreover, a second doctor needs to check and sign the same report (four-eyes principle). In this example the role *doctor* is equipped with both permissions, i.e., filing and signing a report. To prevent a single subject from performing both tasks on the same report (thus undermining the four-eyes principle) we have to constrain these two tasks with an entailment constraint. *Entailment constraints* (see, e.g., [3,18,20]) provide means for placing restrictions on the subjects who can perform a task x given that a certain subject has performed another task y. *Mutual exclusion* and binding constraints are typical examples for entailment constraints. For instance, a *dynamic mutual exclusion* (DME) constraint defines that two subjects must not perform two mutually exclusive tasks in the same instance of a Web document. This means, that the permissions to perform two DME tasks can be assigned to the same subject or role, but for each instance of a particular Web document, we need two distinct individuals to perform both tasks. Binding constraints, on the other hand, can be seen as the opposite of mutual exclusion constraints. For example, *subject binding* defines that the subject who performed the first task must also perform the bound tasks.

Ideally, realizing form-based business applications with a real-time collaborative Web application approach would enable us to enforce RBAC and entailment constraints directly as the users collaboratively work on the forms, i.e., by constraining (e.g. by disabling, locking, or hiding) certain control elements in the user interfaces (UI) for certain subjects. However, so far this topic has – to the best of our knowledge – not been addressed in the existing literature. Major open challenges in this context are how to support changes of the configuration of access constrained UI elements at runtime, realizing acceptable performance and update behaviour, and the easy integration with existing Web applications.

In this paper, we address these challenges that are inherent to enforcing access control constraints in the context of real-time collaborative Web applications. The client-side part of our approach follows the Model-View-ViewModel pattern [15]. Additional server-side components complement our service-based architecture. The resulting architecture enables us to support runtime changes and facilitates the integration our approach with existing applications (see Section 6.2). Furthermore, we show that the approach provides acceptable update behaviour and requires only a small performance overhead for the access control tasks. In our experiments, it shows linear scalability (see Section 6.1). The remainder of this paper is structured as follows: An example scenario motivates our approach in Section 2. In Sections 3 and 4 we propose a novel approach supporting constrained and customized UI views. In Section 5, we describe a prototypical implementation and revisit the motivating example. After comparing to related work in Section 7 we conclude in Section 8.

2 Motivating Example and Challenges

As a motivating example, consider a Web-based application where patient health records are maintained using forms for data entry. The data entry procedure is typically included in a business process with well-defined roles and responsibilities (see, e.g., [9]). In previous work, we presented *CoCoForm* [7], a real-time collaborative Web application framework in which several users can concurrently fill out HTML forms.

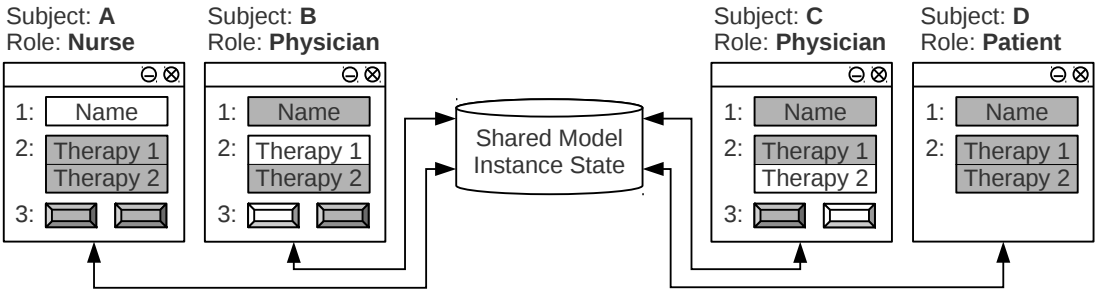


Figure 1: Form-based Collaborative Web Application with Customized Views

Figure 1 shows a simplified example of using *CoCoForm* in the e-health domain. It includes four subjects with shared access to the health record of a patient. The subjects take different roles (nurse, physician, patient) which define their permissions within the application. The nurse enters the name and other personal data of the patient into a textfield (identified by "1"), physician B adds "Therapy 1" to the list of therapies (field "2"), and physician C suggests an additional specialized therapy "Therapy 2". The entire form record is then confirmed by both physicians (buttons "3"). To enforce the four-eyes principle (DME constraint), after physician B clicks the first submit button, the second button is deactivated for physician B, but remains active for physician C. Moreover, each physician can only modify his own therapy suggestions (subject-binding constraint). Finally, the patient should have read-only access to the data. To enforce these constraints, each user has a customized view with partial access to the collaboratively shared model. In Figure 1, white elements can be accessed and modified by the respective user, whereas elements with gray background are subject to access limitations (e.g., read-only but not editable).

A major challenge to realize such customized views for access control constraints is that the *configuration of constrained UI elements* must be *computed server-side* and *effected client-side*. Moreover, this *configuration* might *change dynamically at runtime*. Other challenges are related to *performance and update behaviour*: This means, we immediately need to deliver customized views to all UIs that access the same instance of a Web document (e.g., in the example the UIs need to be updated immediately after one of the subjects changes a document). Such an *immediate update* is required to prevent users from performing actions that were either already performed by another user or that are con-

strained by an entailment constraint (which may have a direct impact on the subjects who are allowed to fill in certain form field for example, see Section 1). In order to be applicable in real-world application scenarios, the approach should *efficiently handle large numbers of simultaneously connected users*. Finally, the approach should allow for an *easy integration with existing Web applications*.

3 Approach Synopsis

The aim of our approach is to support access control and customized views in real-time collaborative Web applications. The *View* of a Web application represents the UI with all visible and invisible elements, form input fields, interactive content, and more. The elements and associated interactions in the UI are subject to constraints (e.g., actions that require a certain permission) which are encoded in well-defined (RBAC) models. Our approach maps the model elements to configuration properties, and clients request the runtime values of these configurations from a *View Service*. The user-specific configurations computed by the server-side *View Service* are then applied to the *View* on the client-side.

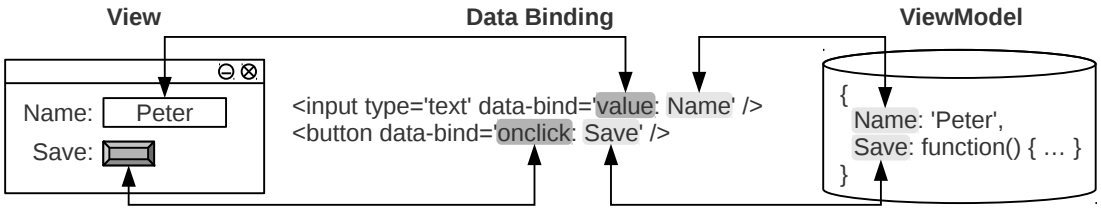


Figure 2: Data Binding between View and ViewModel

As the basic binding concept between the *View* and the *Model*, our approach applies the *Model-View-ViewModel* (MVVM) pattern [15]. The MVVM is a specific version of the Presentation Model pattern (see [6]). It relies on the data binding concept, which ensures that the *View* and the state of its components are bound to properties of a *ViewModel*. This means that changes of the *ViewModel* are automatically reflected in the *View*. For instance, in Figure 2 we can see that the `value` attribute of the `<input>` field is bound to the property `Name` in the *ViewModel*. Secondly, the `onclick` handler of the `button` is bound to the *ViewModel*'s `Save` property. In general, the *ViewModel* acts as a mediator between the Model and the View by encapsulating all logic (e.g., formatting and data type conversion) needed to expose the properties and functionalities of the *Model* to the bound *View*. Additionally, it is in charge of reacting to user commands (e.g., a user fills out an input field) and reflecting them by performing the corresponding *Model* state changes. In general, the MVVM pattern makes it easy to realize the client-side part of the required *View Customization* functionality. In particular, we can customize a client's *View* just by configuring its *ViewModel* properties.

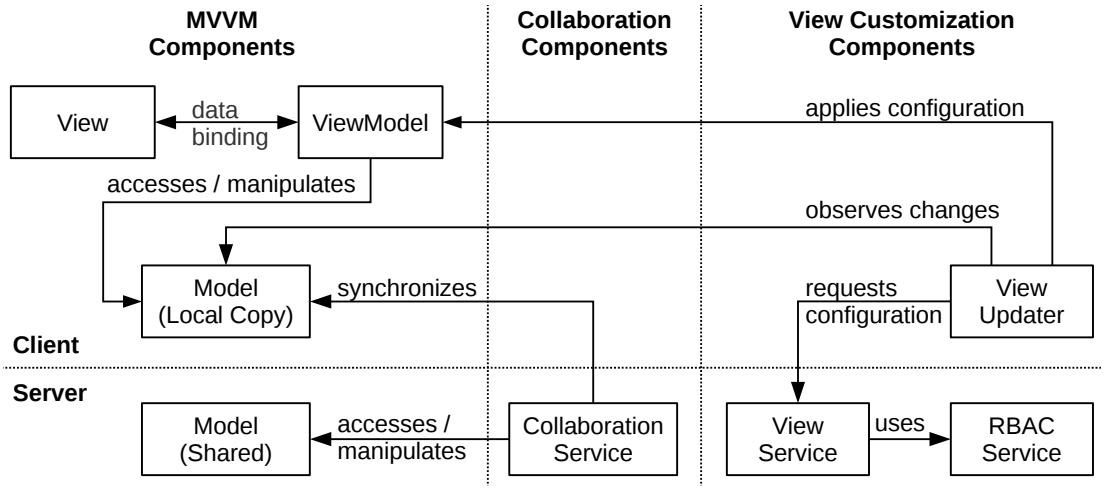


Figure 3: Architectural Overview

Figure 3 provides an architectural overview of the components (i.e., both server-side and client-side) and interactions in our approach that are needed to realize the required *View Customization* functionality. The left-hand column of the figure depicts the core components of the MVVM architecture. In contrast to the classic MVVM architecture, in our approach the *ViewModel* does not directly access/manipulate the shared *Model* (i.e., the shared application state). Instead, it accesses/manipulates only a local copy of the shared *Model*. That is, a *Collaboration Service*, which is the cornerstone of a real-time collaborative Web application, ensures that the server-side shared *Model* is constantly kept in sync with all client-side copies of it. While the *Collaboration Service* allows us to let users collaboratively work on the same Web document, it certainly does not provide means for constraining (e.g., disabling, locking, or hiding) certain control elements in the UI for certain users. Consequently, the *View Service* uses the central *RBAC Service* to compute *ViewModel* configurations. Although these *ViewModel* configurations are computed server-side, they need to be effected client-side, i.e., to constrain UI elements in the *Views* of each client. To account for this, the client-side *View Updater* component of each client actively requests (i.e., pulls) the computed *ViewModel* configurations from the *View Service*. Eventually, these configurations are then applied to the *ViewModel*, which in turn – through data binding – effectively constrain the *Views* of each client.

4 Supporting Customized Views

This section details how the different components of the architecture outlined in Figure 3 enable us to enforce access control policies and entailment constraints directly as the users collaboratively work on a shared *Model*, i.e., by constraining certain control elements in the UI for certain subjects.

Firstly, we want to exemplify our UI customization approach using Figure 4. The figure is divided in two parts, the client-side part and the server-side part.

The figure shows that the *Model* contains only a single property `Name` which is mapped ① to both, a `value` and a `label` property in the *ViewModel*. Next, by applying the basic MVVM pattern, the two properties are bound ② to concrete `<label>` and `<input>` HTML elements in the *View*.

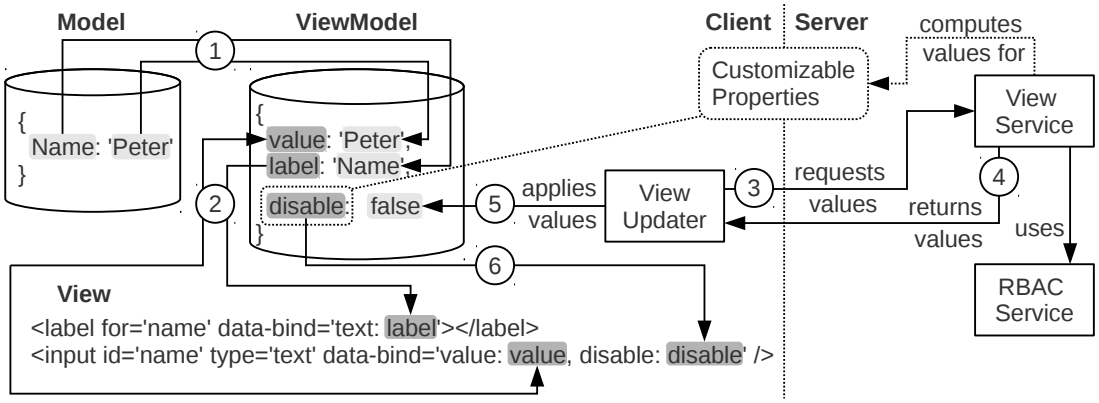


Figure 4: View Customization Example

Next, we assume that the `<input>` field (and the associated action in the RBAC model) is constrained by some RBAC policy. To customize the `<input>` and dynamically make it enabled or disabled, we add the `disable` property to our *ViewModel*. The name of the property (`disable`) is added to the set of *Customizable Properties*. That is, we do not want the client to decide about the value of the `disable` property on its own. Instead, the server has to compute the values for each *Customizable Property*. Thus, the client requests ③ the values from the server-side View Service. The View Service uses the RBAC Service to determine the concrete value for the `disable` property (`true` if and only if the client is allowed to change the `Name` property of the *Model*). The View Service returns ④ the list of *Customizable Properties* together with their customized values to the client-side View Updater. Next, the View Updater applies ⑤ these customized values to the *ViewModel*. Finally, the property value is automatically reflected ⑥ in the *View*, as we have bound the `disable` property of the *ViewModel* to the `disabled` flag of our `<input>` field.

Abstracting from the example in Figure 4, the basic idea of our approach is that the core *ViewModel* is augmented with additional *Customizable Properties*. These properties are used to easily implement customizations in the *View* (e.g., enabling/disabling an `<input>` field). While the property names are defined and processed on the client-side, the actual values for these properties are computed for each user separately on the server side. In summary, the purpose of the *Customizable Properties* is twofold:

1. **Enablement.** At the client-side, these properties have an enabling character, i.e. they allow for realizing the customization of the *View*.
2. **Contract.** Additionally, they can be considered as a contract between the *ViewModel* and the server-side View Service. That is, the client-side *View-*

Model defines the set of *Customizable Properties* and the server-side View Service provides the actual values for these properties. For instance, if the server returns a value of `true` for the `disable` property (see the example above), the client is responsible for actually disabling the `<input>` field in the client's *View*. Hence, the client and the server must have a common view of the semantics of each property.

4.1 Client-side Updates of the ViewModel

The View Updater is in charge of requesting and applying *ViewModel* configurations from the View Service. We propose a simple request/response style of communication between these two components.

```
1  var subject, role,
2      viewModel = {
3          value: 'Peter', label: 'Name', // core properties
4          disable: false, visible: true   // customizable properties
5      };
6
7  function requestView() {
8      var xhr = new XMLHttpRequest(),
9          uri = '/viewService?subject=' + encodeURI(subject) + '&role=' + encodeURI(role);
10     xhr.open('GET', uri);
11     xhr.onload = function() {
12         var configuration = JSON.parse(this.response); // e.g. {disable: true, visible: true}
13         for (var property in configuration) {
14             viewModel[property] = configuration[property];
15         }
16     };
17     xhr.send();
18 };
19
20 function onModelChange(property, value) { // called whenever the Model changes
21     requestView();
22     viewModel[property] = value;
23 };
```

Listing 1: A Simple View Updater Example

Listing 1 illustrates an excerpt of the corresponding exemplary client-side JavaScript code. After firing the request (line 17) we asynchronously process the response that contains the requested *ViewModel* configuration. In the example from Listing 1, the *Customizable Properties* consist of two properties `disable` and `visible` (line 4). Correspondingly, the *ViewModel* configuration returned by the View Service contains concrete values for these two properties, e.g., `{disable: true, visible: true}`. The next step is to apply this configuration to our *ViewModel*. To this end, the JSON-encoded result of the View Service is parsed, and each entry in the result is applied to the local `viewModel` variable (lines 12-15).

Having discussed how the View Updater requests and applies *ViewModel* configurations, we now draw our attention to the question when it should issue its requests. In general, we can say that this depends on the application's context.

However, in our context, i.e., RBAC and entailment constraints, we can also say that *Views* need to be updated exclusively after a *Model* change has happened. Whenever a property is changed in the shared *Model* (i.e., the application state), all *Views* need to be re-computed and (potentially) updated. This circumstance is also reflected in Listing 1 (lines 20-23), where we can see that a new request is triggered for every *Model* change that happens (via the `onModelSync()` callback).

4.2 Server-side Computation of ViewModel Configurations

The computation of *ViewModel* configurations is done server-side, i.e., by the View Service. Upon a request, the View Service returns a *ViewModel* configuration to the requesting client-side View Updater component.

```
1  function onRequest(subject, role) {
2      var property = 'Name', // there is just a single 'Name' property in our model
3          response = {
4              disable: !rbacService.canWrite(subject, role, property),
5              visible: rbacService.canRead(subject, role, property)
6          };
7      return response; // e.g. {disable: false, visible: true}
8  }
```

Listing 2: Basic View Service Example

For instance, in Listing 2 we can see an excerpt of the implementation of a very basic View Service[4] that is tailored to return a configuration for the set of *Customizable Properties* defined in the application code presented in Listing 1. In essence, the service has to compute values for the two *Customizable Properties*, i.e., `disable` and `visible`. As we can see (line 4), it "asks" the central RBAC Service if the provided `subject`/`role` combination has the permission to change (i.e., write) the application's *Model* property, i.e., `Name`. A positive answer (i.e., the user has the permission to change the *Model* property) is reflected with a `disable` value of `false`, which in turn enables the UI element and eventually allows this specific user to manipulate the *Model* property in her customized *View*. Similarly, the service uses the RBAC Service to determine a value for the `visible` property. Eventually, it returns the JSON-encoded configuration (line 7) to the requesting client. Note, that the required parameters of the service, i.e., `subject` and `role` could be supplied as URI parameters (as in line 9 in Listing 1).

5 Implementation – The CoCoForm Framework

This section discusses a prototype implementation of our approach, called Constrainable Collaborative Forms (CoCoForm)[5]. We used CoCoForm to implement and evaluate the e-health record case from Section 2.

[4] Note that we chose JavaScript solely for its well-known and concise syntax.

[5] A proof-of-concept demo is available at `http://demo.swa.univie.ac.at/cocoform2`

Our prototype is based on the OpenCoweb[6] framework, which consists of both, a Collaboration Service (as in Figure 3) and a (client-side) JavaScript API. The latter allows to subscribe to incoming *Model* change events, i.e., by registering a callback function which in turn enables us to trigger our View Updater component (as in Listing 1).

The View Updater issues simple XMLHttpRequests to obtain *ViewModel* configurations from the View Service. The View Service is implemented as a plain HTTP Service in Java, using the JAX-RS API[7], and the configurations are returned in JSON format. The central RBAC Service, which is utilized by the View Service, has been presented in previous work [7]. We use a model-driven approach for defining forms and securing them using access control constraints. Server-side we internally work with Ecore[8] model instances which are marshalled into JSON for the client-side JavaScript application.

Besides OpenCoweb's JavaScript API, we use the Knockout[9] library for realizing the MVVM pattern in the client-side application code. In particular, we also use Knockout's Mapping plugin which allows us to automatically transform the JSON-encoded *Model* into a *ViewModel*. The Mapping plugin also allows us to easily update the *ViewModel* whenever the *Model* changes. Additionally, we augment the *ViewModel* with additional `visible` and `editable` properties. We also use Knockout's template mechanism to (1) create the needed input fields and buttons on-the-fly and (2) establish data binding using corresponding `data-bind` attributes.

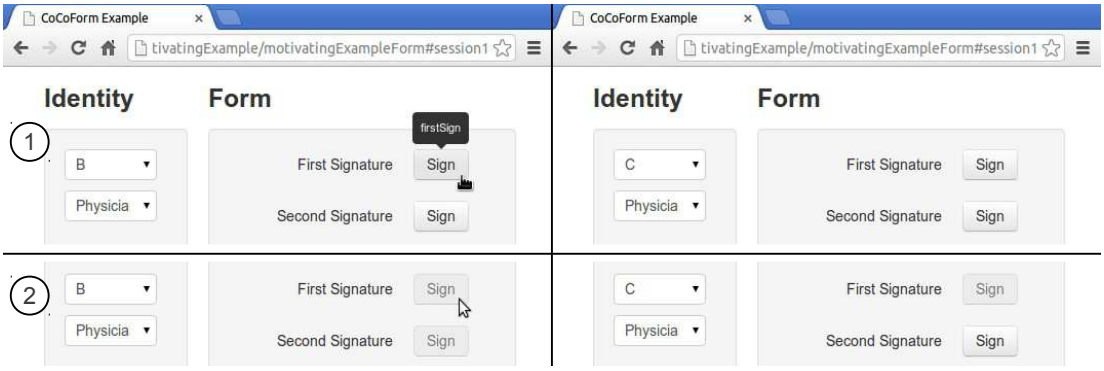


Figure 5: Customized Views and Dynamic Mutual Exclusion with *CoCoForm*

Motivating Example Revisited Now we want to revisit the dynamic mutual exclusion example from Section 2 and discuss a concrete implementation using *CoCoForm*. Figure 5 shows four screenshot excerpts of an example form with

[6] OpenCoweb, `http://opencoweb.org`
[7] JAX-RS, `http://jax-rs-spec.java.net`
[8] Eclipse Modeling Framework, `http://www.eclipse.org/modeling/emf`
[9] Knockout, `http://knockoutjs.com`

two dynamically mutual exclusive buttons. In particular, these buttons represent the first and the second signature on a patient record (as described in Section 2). Figure 5 is vertically split into two columns, i.e., the *View* of the first user (subject B) and the second user (subject C). Both subjects are concurrently working on this form. In the first row (indicated with ①) we can see that both buttons are available for both subjects. The mouse pointer in the upper left part indicates that subject B clicks the first signature button. This click results in a *Model* change which triggers the View Updater component of both clients. As a result, the View Updaters of both clients issue a request to the View Service, resulting in the updated Views in ②. While the first button has been disabled for both clients (which reflects the requirement that any form element can only be manipulated once), the second button is only disabled for subject B. This is due to the dynamic mutual exclusion constraint which demands that subject B, who has just clicked the first button, must be prevented from clicking the second button (see Section 2). However, subject C is still allowed to click the second button. In summary, this example illustrates how our approach enforces access control constraints in real-time collaborative Web applications by dynamically changing the UIs of each user at runtime.

6 Evaluation

In the following sections we discuss both, our lessons learned and the limitations of our approach and the findings of the conducted performance evaluation.

6.1 View Service Performance Evaluation

In the context of real-time collaborative Web applications, users typically expect instantaneous update behavior, which led us to study in how far our UI customization approach meets this requirement. We identify the View Service as a potential performance bottleneck. In particular, we anticipate that requests issued by a potentially large number of users (i.e., resulting from a *Model* change) need to be handled concurrently by *CoCoForm's* View Service.

All measurements have been conducted on a machine equipped with a 2.4 GHz dual core CPU, 8 GB RAM, running Ubuntu GNU/Linux 12.10. Both, the View Service and the testing tool, i.e., Apache's `ab` tool[10], ran on the same machine. Hence, the measurements are free from any network-induced effects such as latency, jitter and so on.

Figure 6 depicts the average response times of both, the actual View Service (solid line) and a "Null" (i.e., no computation at all) Service (dashed line), for a given number of concurrent requests. For instance, in the case of 600 concurrent requests, the average response time for all clients is roughly 200 ms while the response time of the Null Service is roughly 50 ms. This means, that in this case it takes roughly 150 ms to compute a single *ViewModel* configuration, while the rest of 50 ms accounts for the underlying communication and Web Service stack.

[10] Apache `ab` tool, `http://httpd.apache.org/docs/2.4/programs/ab.html`

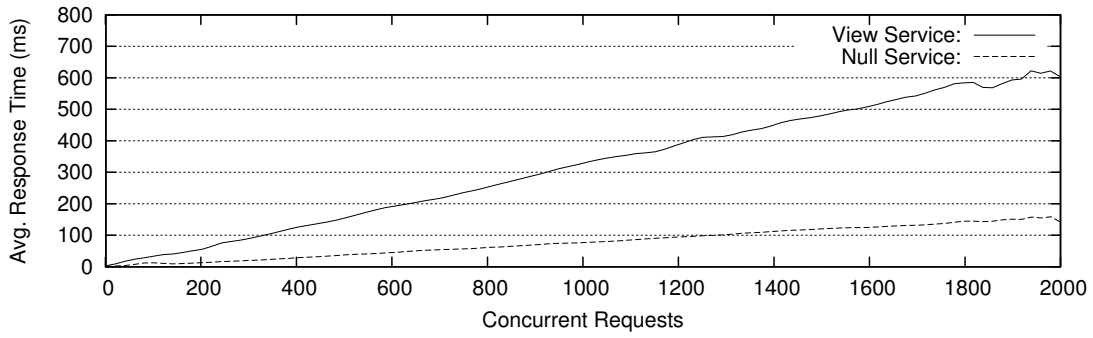


Figure 6: View Service Response Times

The evaluation results indicate that our View Service implementation has linear scalability. Even in the case of 2000 users working on the same form document collaboratively, the average response time remains well below a second. In our experiment, the View Service's response times amount to approximately four times the response times of the Null Service. As the Null Service represents the theoretical minimum that is possible for the given Web Service framework, we consider the performance overhead acceptable.

6.2 Lessons Learned

We implemented the *CoCoForm* prototype (see Section 5) to demonstrate the feasibility of our approach (see Section 4). We showed that access control policies and entailment constraints in the context of real-time collaborative Web applications can effectively be enforced by dynamically constraining UI elements for certain subjects. In the following paragraphs we want to discuss our lessons learned and the limitations of our approach.

Our approach is complementary to currently available frameworks and solutions that support the development of real-time collaborative Web applications such as Apache Wave[11], ShareJS[12] and OpenCoweb (see Section 5). This is due to the fact that it is completely decoupled from the collaborative aspects of the application. In essence, supporting customized views using our approach merely requires the deployment of a single, dedicated and self-contained View Service as well as hooking-in the View Updater code into the client-side application code.

Although our approach is built upon the MVVM pattern, it does not exclude other approaches (e.g., the classic Model-View-Controller pattern). Instead, we argue that our approach can coexist with others. In that case, the *ViewModel* is solely used to realize the customizable parts of the *View*. Hence, it just contains the set of *Customizable Properties*. The only requirement is that the corresponding DOM nodes (e.g., `<input>` elements) are augmented with additional data binding attributes (e.g., `data-bind`). Note that this even works in the case of

[11] Apache Wave, `http://incubator.apache.org/wave`
[12] ShareJS, `http://sharejs.org`

dynamically generated (i.e., generated using JavaScript code) DOM nodes, as long as it is possible to add the data binding attributes.

A major concern – especially in the context of real-time collaborative Web applications – is the ability to apply the View customization nearly instantaneously. In other words, the response times of the View Service must be kept low. Keeping the response time low with a growing number of simultaneously connected users, requires that the system is able to scale. Our View Service itself is completely stateless, as (1) each request contains all necessary information (e.g., subject and role) that is needed to compute a *ViewModel* configuration and (2) no information at all needs to be persisted. This stateless nature as well as the simple request/response style of communication between the View Updater and the View Service allows for scaling horizontally in a straightforward manner, i.e., the communication can be routed through a load-balancing proxy that distributes each request among multiple instances of the service.

However, the request/response communication style also comes with a couple of challenges. For example, there is the issue of "the needless request". This is the case when the View Service returns a *ViewModel* configuration that is not different from the currently active one. Hence, we could have saved both client-side and server-side computing resources (e.g., CPU time, network bandwidth, etc.) if we simply had not issued this "needless request" in the first place. This issue can be addressed using a push approach (instead of the presented pull approach). That is, the View Service would selectively push new *ViewModel* configurations to the clients only if it is necessary (i.e., at least one *ViewModel* property needs to be changed). However, this push approach introduces a certain amount of complexity to the View Service. For instance, it would require an explicit session handling, i.e., in a push scheme we have to maintain a list of connected clients to correctly update the corresponding *ViewModels*. Moreover, a push scheme would also require to keep track of each client's *ViewModel* to determine if we need to push a new *ViewModel* to a particular client. In summary, the push approach allows for avoiding "needless requests" (in fact, no requests are made at all) while the pull approach comes with a lower complexity, especially when scaling (i.e., when multiple instance of the View Service have to coordinate session with each client's *ViewModel* configuration). Another idea to – at least – mitigate this problem would be a more efficient client-side triggering logic. For instance, we could provide the clients with a list of *Model* properties that are not constrained by any access control constraint at all. Then, the clients would not need to request a new *ViewModel* configuration whenever a *Model* change event arrives that is contained in the list of unconstrained properties.

In our approach access control policies and entailment constraints are enforced client-side, i.e., by constraining UI elements. From a security perspective, however, we often cannot trust code that is executed on the client (i.e., the browser). The reason is that we can not prevent a potential attacker from modifying the code to be executed. For instance, an attacker might be able to change the *ViewModel* configuration to gain access to a constrained UI element and eventually pass a *Model* change event (i.e., concerning a constrained *Model*

property) to the Collaboration Service. However, we could contain the effects of such client-side code injections by preventing such unauthorized *Model* changes (1) from being applied to the server-side *Model* and (2) from being distributed to other session participants. This can be achieved by routing all incoming (i.e., coming from the clients) *Model* change events trough an enforcement proxy. This proxy uses the RBAC Service to decide if it should forward the event to the Collaboration Service (i.e., in case the client has the permission to change the *Model* property) or not. This guarantees that client-side code injections do not lead to server-side *Model* changes or impact session participants.

Finally, our approach assumes that the *Model* is being synchronized with all clients. That is, all clients "see" exactly the same *Model*. However, if this *Model* contains sensitive information, this might be an issue. We will address this problem as part of our ongoing research.

7 Related Work

In this section we discuss related work in the area of customized and shared application views, collaboration platforms as well as access control enforcement.

Customized and Shared Application Views. Similar to customized views in our approach, Koidl et al. [12] propose user-specific Web site rendering. However, their approach aims at user-centric personalization of Web experience, whereas the customized views in our approach result from RBAC policies and entailment constraints. An interesting aspect in their solution is that the personalization is cross-site, i.e., it spans the Web sites of multiple providers. Our approach currently does not implement cross-provider policies. However, we presented a related approach for cross-organizational access control in Web service based business processes in [9]. As part of our future work, we will integrate cross-site capabilities in our approach for real-time collaborative Web applications. Berry et al. [2] have applied role-based view control to desktop applications. Their approach captures the virtual framebuffer of application windows and applies blurring, highlighting, pixelizations, and other manipulations over the rendered view. Our approach benefits from the fact that manipulation of Web user interfaces is easier to achieve; using the path to the target DOM element, our client-side View Updater takes care of customized view manipulations.

Collaboration Platforms. The seminal work of Sun et al. [19] proposes the transparent adaptation (TA) approach to convert single-user applications into collaborative multi-user applications. The cornerstone of TA is operational transformation (OT) [4]. Our approach is orthogonal to OT: the RBAC policies and entailment constraints provide an application workflow with well-defined responsibilities, and we maintain document consistency by allowing only sequences of operations that comply with this workflow. Farwick et al. [5] discuss an architecture for Web-based collaborative metamodeling. Their framework allows multiple users to work on graphical meta-models collaboratively. Modifications of the (meta-)models are secured by basic access control measures, but in contrast to our work, they do not explicitly address customized views and dynamic

updates resulting from the enforcement of RBAC entailment constraints. Heinrich et al. [8] present a generic collaboration infrastructure aimed at transforming existing single-user Web applications into collaborative multi-user Web applications by synchronizing DOM trees. In other words, their approach makes sure that the DOM trees of all clients in a collaborative session is constantly kept in sync. As we strive for customizing the DOM tree for each client, this approach is completely at odds with ours. Consequently, we require synchronization to take place at the model-level instead of the view-level (as in [8]).

Security and Access Control Enforcement. A plethora of approaches have been presented for integrating security and access control in Web applications. Joshi et al. [10] provide an early study on generic security models for Web-based applications. Starnberger et al. [16] use smart card based security and discuss a generic proxy architecture to enforce authorizations. In [1], Belchior and colleagues model RBAC policies using RDF triples and N3Logic rules. Mallouli et al. [13] use extended finite state machines (EFSM) to model systems with OrBAC [11] (Organization Based Access Control) security policies. However, none of these approaches addresses the enforcement of access control policies and entailment constraints in dynamic real-time Web applications.

8 Conclusion and Future Work

In this paper, we demonstrate that access control policies and constraints – in particular entailment constraints – in the context of real-time collaborative Web applications can effectively be enforced by dynamically constraining UI elements for certain subjects. We show that our service-based approach can be used to realize the corresponding UI view configuration functionality and we provide evidence that it is potentially capable of meeting the – especially in the context of real-time collaborative Web applications important – requirement of nearly instantaneous update behavior, even for a large number of simultaneously connected users. Although the client-side part of the UI view configuration functionality is built upon the MVVM pattern, we show that it can easily coexist with others.

As future work we will look into privacy issues (see Section 6.2) and apply our approach to other types of collaborative processes. In particular, we are interested in establishing the concept of entailment constraints in more dynamic processes (e.g., text editing or modeling) where we will have to deal with completely dynamic (i.e., changing at runtime) access control and constraint models.

References

1. Belchior, M., Schwabe, D., Silva Parreiras, F.: Role-based access control for model-driven web applications. In: 12th International Conference on Web Engineering (ICWE). pp. 106–120 (2012)
2. Berry, L., Bartram, L., Booth, K.S.: Role-based control of shared application views. In: 18th ACM symposium on User interface software and technology (UIST). pp. 23–32 (2005)

3. Bertino, E., Ferraria, E., Atluri, V.: The specification and enforcement of authorization constraints in workflow management systems. ACM Transactions on Information and System Security 2(1), 65–104 (1999)
4. Ellis, C.A., Gibbs, S.J.: Concurrency control in groupware systems. SIGMOD Record 18(2), 399–407 (1989)
5. Farwick, M., Agreiter, B., White, J., Forster, S., Lanzanasto, N., Breu, R.: A web-based collaborative metamodeling environment with secure remote model access. In: 10th International Conference on Web Engineering (ICWE). pp. 278–291 (2010)
6. Fowler, M.: Presentation model. Essay, July (2004)
7. Gaubatz, P., Zdun, U.: Supporting entailment constraints in the context of collaborative web applications. In: 28th Symposium On Applied Computing (2013)
8. Heinrich, M., Lehmann, F., Springer, T., Gaedke, M.: Exploiting single-user web applications for shared editing: a generic transformation approach. In: Proceedings of the 21st international conference on World Wide Web. pp. 1057–1066 (2012)
9. Hummer, W., Gaubatz, P., Strembeck, M., Zdun, U., Dustdar, S.: An integrated approach for identity and access management in a SOA context. In: 16th ACM Symposium on Access Control Models and Technologies (SACMAT) (2011)
10. Joshi, J.B.D., Aref, W.G., Ghafoor, A., Spafford, E.H.: Security models for web-based applications. Communications of the ACM 44(2), 38–44 (2001)
11. Kalam, A.A.E., Benferhat, S., Miège, A., Baida, R.E., Cuppens, F., Saurel, C., Balbiani, P., Deswarte, Y., Trouessin, G.: Organization based access control. In: 4th IEEE Int. Workshop on Policies for Distributed Systems and Networks (2003)
12. Koidl, K., Conlan, O., Wade, V.: Towards user-centric cross-site personalisation. In: 11th International Conference on Web Engineering (ICWE). pp. 391–394 (2011)
13. Mallouli, W., Orset, J.M., Cavalli, A., Cuppens, N., Cuppens, F.: A formal approach for testing security rules. In: 12th ACM symposium on Access control models and technologies (SACMAT). pp. 127–132. ACM (2007)
14. Sandhu, R., Coyne, E., Feinstein, H., Youman, C.: Role- based access control models. Computer 29(2), 38 –47 (1996)
15. Smith, J.: WPF apps with the Model-View-ViewModel design pattern. MSDN magazine (2009)
16. Starnberger, G., Froihofer, L., Goeschka, K.M.: A generic proxy for secure smart card-enabled web applications. In: 10th International Conference on Web Engineering (ICWE). pp. 370–384 (2010)
17. Strembeck, M.: Scenario-driven Role Engineering. IEEE Security & Privacy 8(1) (January/February 2010)
18. Strembeck, M., Mendling, J.: Generic algorithms for consistency checking of mutual-exclusion and binding constraints in a business process context. In: On the Move to Meaningful Internet Systems (OTM). pp. 204–221 (2010)
19. Sun, C., Xia, S., Sun, D., Chen, D., Shen, H., Cai, W.: Transparent adaptation of single-user applications for multi-user real-time collaboration. ACM Transactions on Computer-Human Interaction 13(4), 531–582 (2006)
20. Wainer, J., Barthelmes, P., Kumar, A.: W-RBAC - A Workflow Security Model Incorporating Controlled Overriding of Constraints. International Journal of Cooperative Information Systems (IJCIS) 12(4) (Dec 2003)

Paper G

Enforcing Entailment Constraints in Offline Editing Scenarios for Real-time Collaborative Web Documents

The subsequent paper has been published as follows:

P. Gaubatz, W. Hummer, U. Zdun, and M. Strembeck. Enforcing Entailment Constraints in Offline Editing Scenarios for Real-time Collaborative Web Documents. In *29th Symposium On Applied Computing*, Gyeongju, Korea, March 2014.

This paper further extends the concepts presented in Paper E and Paper F. In particular, we identified a potentially weak spot of existing solutions for enforcing access control restrictions in service-based systems (including our own previously presented work). Namely, they typically rely on a central service, the policy decision point. However, for use cases with unreliable or limited connectivity, such as mobile devices, a permanent connection to this centralized policy decision point can not be guaranteed. Therefore, we tackled this problem by proposing a novel approach that includes methods for client-side enforcement of access control constraints for offline users, and merging of offline changes, that enables users to edit such access constrained shared documents offline. The proposed approach includes a generic conflict detection and resolution approach that attempts to resolve merge conflicts that are inherent to access constrained documents automatically while prioritizing online users and maximizing the number of filled out data fields in a document. Eventually, we substantially extended our CoCoForm prototype implementation and conducted extensive performance evaluations.

Enforcing Entailment Constraints in Offline Editing Scenarios for Real-time Collaborative Web Documents

Patrick Gaubatz[1], Waldemar Hummer[2], Uwe Zdun[1] and Mark Strembeck[3]

[1]Faculty of Computer Science
University of Vienna
{first.last}@univie.ac.at

[2]Distributed Systems Group
Vienna University of Technology
hummer@infosys.tuwien.ac.at

[3]Institute for Information Systems
WU Vienna
mark.strembeck@wu.ac.at

ABSTRACT

Real-time collaborative Web applications allow a multitude of users to concurrently work on a shared document. Especially in business contexts it is often necessary to be able to precisely define and restrict who is allowed to edit which data field of such a shared document. Existing solutions for enforcing such access control restrictions typically rely on a central service, the policy decision point. However, for use cases with unreliable or limited connectivity, such as mobile devices, a permanent connection to this centralized policy decision point can not be guaranteed. To address this problem, we present a novel approach that includes methods for client-side enforcement of access control constraints for offline users, and merging of offline changes, that enables users to edit such access constrained shared documents offline. We propose a generic conflict detection and resolution approach that attempts to resolve merge conflicts that are inherent to access constrained documents automatically while prioritizing online users and maximizing the number of filled out data fields in a document. In addition, we discuss and evaluate our approach via a prototype implementation.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Web Applications; D.2.2 [**Software Engineering**]: Design Tools and Techniques

Keywords

Document Merge, Conflict Detection, Conflict Resolution, Authorization, Access Control Enforcement

1. INTRODUCTION

Real-time collaborative Web applications such as Google Docs or Etherpad aim to efficiently support the joint work of team members, allowing them to collaboratively work on the same Web document at the same time. While such text based applications are probably the most popular example of collaborative Web applications today, their use in typical business contexts is limited as many business applications usually deal with strict, standardized forms with precisely specified text fields (see, e.g., [11]). Another crucial aspect of business applications is access control. The general goal of our work is to address this combination of form-based collaborative Web applications with access control; in this paper, we specifically focus on offline editing of real-time collaborative Web forms which have access control constraints.

Role-based access control (RBAC) [17] is the de-facto standard for access control in software systems. In RBAC, roles are used to model different job positions and scopes of duty within a system. These roles are equipped with permissions to perform tasks. Human users (subjects) are assigned to roles according to their work profile. For example, in a real-time collaborative Web document for the e-health domain only subjects assigned to the role *physician* shall be allowed to prescribe medications by filling in the corresponding text field of a document. To enforce a four-eyes principle, a second *physician* shall check and sign these prescriptions. In this example the role *physician* is equipped with both permissions, i.e., prescribing and signing medications. To prevent a single subject from performing both tasks and undermining the four-eyes principle we have to constrain these two tasks. Entailment constraints (see, e.g., [6, 20, 22]) provide means for placing such restrictions on the subjects who can perform a task x given that a certain subject has performed another task y. Mutual exclusion and binding constraints are typical examples for entailment constraints. For instance, a dynamic mutual exclusion constraint can be used to realize the four-eyes principle. In particular, it defines that two subjects must not perform two mutually exclusive tasks of the same Web document.

The decision if a specific subject should be granted the permission to perform a task (e.g, filling in a field) is called authorization decision. In the context of Web-based environments, authorization decisions are typically delegated to a central service, the Policy Decision Point (PDP) (see, e.g., [14]). As a consequence, using a Web application that is subject to access control usually requires a user's device to have reliable network connectivity (to be able to access the PDP). However, the increasing importance of mobile computing and mobile devices in the context of business applications demands solutions that enable users to continue editing (entailment constrained) collaborative Web documents even if a reliable network connection can not be guaranteed. Such unreliable network connections frequently occur if the user

SAC'14 March 24-28, 2014, Gyeongju, Korea.

is on an airplane, in a train, in a basement, or in a rural area. To the best of our knowledge, existing literature does neither consider the enforcement of entailment constraints for offline editing of collaborative Web documents, nor does it provide a systematic approach for merging actions that have been performed offline on a shared collaborative Web document that is subject to entailment constraints.

In this paper, we therefore propose a novel approach to support access control enforcement for offline users. In our approach, the offline changes on a Web document are authorized and recorded at the client-side. After re-establishing a network connection the corresponding changes are merged with the online, collaborative Web document. To handle potential violations of entailment constraints that may result from merging changes that were performed offline, our approach includes automated conflict detection and resolution procedures. In addition, our approach allows for prioritizing online performed changes (over changes that were performed offline) and maximizes the number of filled out data fields in a Web document.

The main contributions of this paper are as follows:

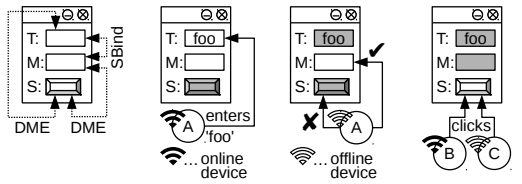
- We propose a system architecture that enables users of real-time collaborative Web applications to edit entailment constrained Web documents offline.
- We discuss how merge conflicts are inherent to editing such constrained Web documents offline and present a merge approach capable of detecting and resolving these conflicts automatically.
- We explain how the approach can be mapped to already existing Web browser technologies and APIs.
- We discuss performance and scalability measurements of the implemented prototype and evaluate the degree of automation of the proposed merge approaches.

The structure of this paper is as follows: Section 2 motivates our approach using an example scenario. Section 3 presents the conceptual details of our approach. We discuss our prototype implementation in Section 4 and evaluate the approach in Section 5. After a discussion of related work in Section 6, we conclude in Section 7.

2. MOTIVATING SCENARIO

For illustration, we consider a collaborative Web application for maintaining patient health records. Figure 1a shows a simplified form with two text fields (T and M) and a button (S). A *physician* is supposed to enter a therapy plan into text field T. Text field M is used to prescribe a specific medication. The hospital requires that only the *physician* who originally proposed a therapy plan is allowed to prescribe the medication. Entailment constraints (see, e.g., [6, 20, 22]) provide means for defining such restrictions. In this particular example, a subject-binding (SBind) constraint defines that the subject entering text field M must be the *same* that has filled in text field T. Moreover, to enforce a four-eyes principle, the hospital also requires prescriptions to be confirmed (button S) by a *different* physician. Hence, a dynamic mutual exclusion (DME) constraint is used to define that the subject filling in text field M must not be allowed to click button S.

We now consider subjects A, B and C who are concurrently editing this form. Figure 1b shows that Subject A,



(a) Constraints (b) Enforcement (c) Offline Enf. (d) Conflicts

Figure 1: Exemplary Collaborative Web Application

whose device is currently online, fills in field T. As a result, the system must disable text field S for Subject A (indicated by a gray background) in order to prevent violation of the DME constraint. Now, suppose that Subject A loses its network connection. Ideally, the Web application should allow Subject A to continue working offline, while continuing to enforce the entailment constraints (i.e., by allowing A to edit text field M and preventing A from editing text field S, see Figure 1c) even without a connection to the central PDP. In such a scenario, authorization decisions are (temporarily) made at the client side.

After Subject A has filled out T, both Subject B and C can potentially click S. Assume in Figure 1d that Subject C also (temporarily) loses network connectivity, while B confirms the prescribed medication by clicking S. Since C is offline and can not be informed of this action, C is also (locally) permitted to click the same button. Eventually, as Subject C re-establishes its network connection, the Web application is confronted with conflicting actions, because only a single confirmation can be stored in the document. The naive approach would be to enforce a first-come-first-serve principle by blocking C's action. However, this approach may be suboptimal if the global goal is to fill out as many form fields as possible. Moreover, it may result in violations of potentially defined entailment constraints. Thus, if we consider that C submits an entire set of actions performed offline, it may for instance be advantageous to revert B's changes and give precedence to C's.

In summary, we identify two main challenges:

- If a client loses connectivity, both the authorization decision making and the enforcement of entailment constraints have to be conducted at the client side instead of relying on the central server-side PDP.
- Offline editing of entailment constrained real-time collaborative Web documents may lead to conflicts that must be detected and resolved automatically.

3. OFFLINE EDITING APPROACH

Figure 2 provides an architectural overview of the components and interactions used to support offline editing for an entailment constrained collaborative Web document.

The left-hand column of the figure depicts the core components of a real-time collaborative Web application. A Collaboration Service allows users to concurrently work on the same collaborative Web document by constantly keeping the server-side Shared Model (i.e., the data model/content of a concrete Web document) in sync with all client-side Local Models (i.e., exact copies of the Shared Model).

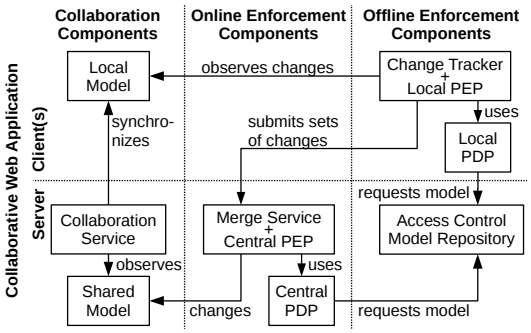


Figure 2: Architectural Overview

To prevent a user from unauthorized manipulation of protected parts of this Shared Model, our approach requires all model changes (i.e., changes of data fields within a Web document) to be routed through a Merge Service which acts as the central Policy Enforcement Point (Central PEP). This service enforces the authorization decision of the Central PDP. More specifically, the Merge Service only applies a change if the Central PDP confirms that the user actually has the permission to perform that change.

Supporting offline editing builds upon three basic ideas:

1. We persist the Local Model and duplicate the functionality of the Central PDP to the clients, allowing them to make authorization decisions locally (see Section 3.1).

2. The tracked changes of the Local Model on the clients are submitted to the central Merge Service as soon as the network connectivity is regained (see Section 3.2).

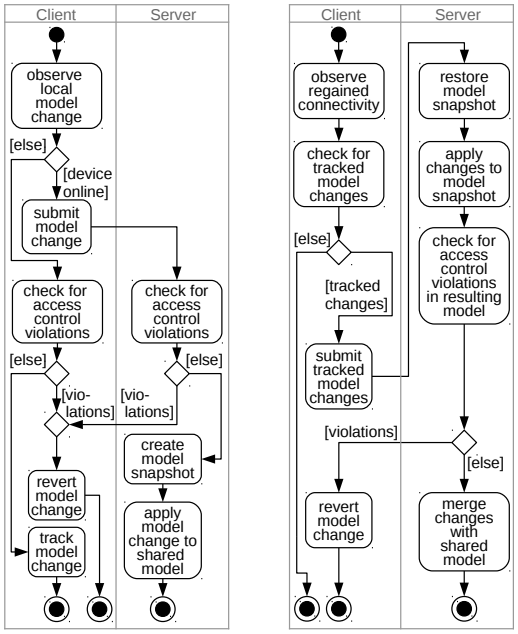
3. We strive to detect and automatically resolve conflicts that result from merging offline performed changes with the Shared Model (see Section 3.3).

3.1 Supporting Client-side Enforcement

Figure 3a depicts the steps that are triggered whenever a user makes changes to their Local Model.

The Change Tracker component observes such changes. If the user is online, the change is directly submitted to the Merge Service, where it is checked for violations of entailment constraints and eventually applied to the Shared Model. In case of violations, the change is discarded and the corresponding data field in the client's Local Model is reverted to its previous state. Otherwise, if the user's device is offline, the Change Tracker requests an authorization decision from the Local PDP to determine if the change violates any constraints, in which case it gets reverted. Otherwise, the change is tracked and stored by the client until a merge with the Shared Model is possible.

The Central PDP and the Local PDP require the same entailment constraint models, which reside on a central Access Control Model Repository service. To guarantee that the user can continue working offline, the Local PDP requests and persists the current model state on client side. Ideally this is done as soon as a specific Web document is accessed for the first time.



(a) Steps that are triggered when a user changes a data field

(b) Steps that are triggered when a user regains network connectivity

Figure 3: Client-side Enforcement and Merging

3.2 Merging Offline Performed Changes

The Change Tracker tracks and stores model changes performed offline. Figure 3b shows the steps that are triggered as soon as a user regains network connectivity. First, the Change Tracker checks if (newly) tracked model changes exist. If no model changes were tracked the process ends. Otherwise, it submits the set of changes to the Merge Service.

Before merging the changes with the Shared Model, the Merge Service must first ensure that all offline changes were legitimate. This additional step is crucial to avoid illegitimate model changes (e.g., in case the Local PDP has been tampered with). Checking the legitimacy of changes in a document requires the Central PDP to be aware of *who* (e.g., Subject x using Role y) has performed which changes on a document. Therefore we propose a snapshot-and-replay approach where the Shared Model is versioned, such that a snapshot of the current state is created for every change. For each incoming merge request the Merge Service restores the corresponding snapshot of the Shared Model that a set of offline changes is based on. Finally, the Merge Service applies ("replays") these offline changes to the snapshot and checks the validity of the resulting model. Thereby, the integrity of the client-side Local PDP can be verified and only legitimate changes are merged with the Shared Model.

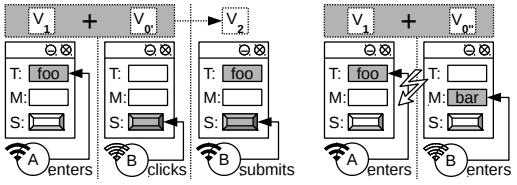
The actual merge procedure involves the Shared Model and a single client's set of offline changes. In essence, we apply each change to the corresponding data field of the Shared Model. To avoid data inconsistencies (if two clients simulta-

Conflict	Description
Duplicate Field	The same field is filled out twice.
Subject-binding (SBind) Violation	SBind constrained fields are filled out by different subjects.
Role-binding (RBind) Violation	RBind constrained fields are filled out using different roles.
Dynamic Mutual Exclusion (DME) Violation	DME constrained fields are filled out by the same subject.
Static Mutual Exclusion (SME) Violation	SME constrained fields are filled out by the same subject or using the same role.

Table 1: Potential Merge Conflicts

neously submit their changes), the Merge Service acquires a lock on the Shared Model. Simultaneously submitted merges are therefore serialized, i.e., processed sequentially.

For illustration, consider the exemplary merge situation depicted in Figure 4a (Figure 4b is discussed in Section 3.3). By filling out text field T, Subject A increases the Shared Model's version counter from V_0 (i.e., the empty document) to V_1. At the same time Subject B, which is offline and still working with V_0, clicks button S. Afterwards, Subject B submits this offline change and eventually (i.e., after restoring V_0, applying the change and validity checking the resulting model) text field T is merged with the current Shared Model's version V_1, resulting in the new merged version V_2.



(a) Non-conflicting Merge (b) Conflicting Merge

Figure 4: Exemplary Merge Situations

3.3 Detecting and Resolving Merge Conflicts

Offline editing for real-time collaborative Web documents may inevitably lead to merge conflicts. Let us reconsider the exemplary merge situation described above (see Figure 4). If we suppose that Subject B fills out text field M while working offline, we are confronted with a merge conflict situation that is depicted in Figure 4b. More specifically, Subject B's offline completed text field M can not be merged without violating the defined subject-binding constraint (see Figure 1a).

In general, a merge conflict happens either when the same field is filled in twice (i.e., online and offline) or when a merge results in a model that violates entailment constraints. We systematically analyzed the respective entailment constraint models (see, e.g., [6,20,22]) to compile a list of conflict types that may arise when merging data fields that are subject to such constraints. Table 1 depicts this set of conflict types.

Detecting merge conflicts is straightforward. However, resolving them (semi-)automatically is not. First of all, there is not a single "one-size-fits-all" resolution strategy (see, e.g., [18]). For instance, to resolve a *Duplicate Field* conflict we could choose between the following resolution strategies: (1) concatenate both (field) values, (2) try to merge both values into a single value, (3) move one value to an attachment and let a human person manually resolve the conflict, or (4) discard one value. However, Strategies 1 and 2 are not always sensible (e.g., if a subject-binding exists), and since Strategy 3 does not resolve conflicts instantaneously we will focus on Strategy 4 for the rest of this paper.

Name	Return Value
getConstraints	Set of constraints a *field* is subject to.
getConstrainedField	Returns the field that is bound to the same *constraint* as *field*.
getSubject	Subject that has filled out *field*.
getRole	Role that has filled out *field*.
violates	`true` if *field* and *anotherField* violate *constraint*.

Table 2: Helper Procedures for Algorithm 2 and 3

The main challenge of this strategy is to decide which values to discard. For instance, in Figure 4b we could discard Subject B's value for field M and leave the Shared Model unchanged. Alternatively, we could revert Subject A's field T and merge Subject B's field M instead. However, by reverting and ultimately deleting Subject A's field partially sacrifice the main idea of the real-time collaboration approach: keeping a document in sync for *online* users that are participating in an *online* collaborative session. Therefore, we allow that changes performed by online users can be prioritized over ones that have been performed by offline users.

Algorithm 1 Basic Merge Algorithm

```
1:  procedure BasicMerge(model, offChanges, offWeight)
2:      fieldsToDelete ← FieldsToDelete(model, offChanges)
3:      if |fieldsToDelete| ≤ |offChanges| × offWeight then
4:          for each fieldToDelete in fieldsToDelete do
5:              model ← model \ fieldToDelete
6:          end for
7:          for each fieldToMerge in offChanges do
8:              model ← model ∪ fieldToMerge
9:          end for
10:     end if
11: end procedure
```

To devise a generic merging approach based on the assumption that prioritization of online changes is crucial in the context of real-time collaboration, we introduce a decision criterion that determines whether a set of online changes should be deleted in order to be able to merge another set of offline changes. This decision is reflected in Algorithm 1. Note, that all required helper procedures are listed in Table 2. In our approach, a set of offline changes ($offChanges$) gets merged only if the set of fields that needs to be deleted ($fieldsToBeDeleted$) is smaller than the former. This approach ensures that a merge—if performed—*never decreases* the number of completed data fields in the Shared Model. To support prioritization of online completed fields we introduce a weighting factor ($offWeight$) that is used to discriminate the set of offline changes (as seen in line 3 of Algorithm 1). For instance, a factor of 0.5 means that removing one online completed field is equally bad as discarding two offline completed fields. Applying this to Figure 4b (i.e., $fieldsToDelete = \{\mathsf{T}\}$ and $offChanges = \{\mathsf{M}\}$) we get $1 \leq 1 \times 0.5$, decide *against merging* and *discard* the given set of offline changes.

Algorithm 1 relies on Algorithm 2 to determine the set of fields that would have to be deleted in order to merge

Algorithm 2 Fields-to-Delete Algorithm

```
1:  procedure FieldsToDelete(model, offChanges)
2:      fieldsToDelete ← ∅
3:      for each field in offChanges do
4:          if field ∈ model then
5:              fieldsToDelete ← fieldsToDelete ∪ field
6:          end if
7:          for each constraint in getConstraints(field) do
8:              otherField ← getConstrainedField(constraint, field)
9:              if otherField ∈ model
                  and violates(field, otherField, constraint) then
10:                 fieldsToDelete ← fieldsToDelete ∪ otherField
11:             end if
12:         end for
13:     end for
14:     return fieldsToDelete
15: end procedure
```

the set of $offChanges$ with the Shared Model. To account for *Duplicate Field* conflicts (see Table 1), Algorithm 2 first adds all fields (contained in $offChanges$) that have already been completed in the Shared Model to the $fieldsToDelete$ list. The remainder of the algorithm (i.e., lines 7–13) deals with conflicts that may occur due to violations of entailment constraints (see Table 1). For instance, regarding Figure 4b this means that merging Subject B's field M, requires deleting Subject A's field T. Otherwise, the subject-binding constraint that is associated with fields T and M (see Figure 1a) would be violated. In general, Algorithm 2 adds all fields to $fieldsToDelete$ that must necessarily be deleted from the Shared Model in order to allow $offChanges$ to be merged without violating any defined entailment constraints.

Algorithm 3 Two-step Merge Algorithm

```
1:  procedure TwoStepMerge(model, offChanges, offWeight)
2:      constrained ← ∅
3:      for each field in offChanges do
4:          if getConstraints(field) ≠ ∅ then
5:              constrained ← constrained ∪ field
6:          end if
7:      end for
8:      unconstrained ← offChanges \ constrained
9:      BasicMerge(model, unconstrained, offWeight)
10:     BasicMerge(model, constrained, offWeight)
11: end procedure
```

Our merge algorithm (i.e., Algorithm 1) realizes an "all or nothing" approach, i.e., the set of offline changes is either completely merged or discarded. For instance, the algorithm might discard a complete set of offline performed changes just because the set contains a single constrained data field that required lots of the Shared Model's data fields to be reverted. Consequently, a key to improving this basic merge approach is to split the set of offline changes up into smaller subsets and merge these subsets with the Shared Model one after another. To this end, we propose a two-step approach (see Algorithm 3) that first tries to merge the distinct set of unconstrained data fields exclusively. Afterwards, the remaining set of constrained data fields is merged. As we show in Section 5 the two-step merge approach exhibits a significantly higher chance that a set of offline changes can be merged than basic merge. However, for large numbers of fields the basic merge approach yields a better performance.

4. IMPLEMENTATION

This section discusses a prototype implementation that has been developed to prove the feasibility of our approach. It is founded on CoCoForm, a Web application framework for real-time collaboration, supporting access control [10,11].

Our approach is completely decoupled from the collaborative aspects of the application. Thus, it is complementary to currently available frameworks for the development of real-time collaborative Web applications, such as ShareJS[1] or the Open Cooperative Web Framework[2] (OpenCoweb). In fact, our prototype extends OpenCoweb, which consists of both, a Collaboration Service (as in Figure 2) and a JavaScript API that allows us to keep the Shared Model in sync with all client-side Local Models. Both, the Change Tracker and the Local PDP issue simple `XMLHttpRequests` to submit local model changes to the central Merge Service and to request entailment constraint models from the Access Control Model Repository. In case of failed requests (i.e., `timeout` or `error` events are emitted) the Change Tracker uses the Local PDP to determine if a change should be tracked or not. We implemented the Local PDP (i.e., in JavaScript) in such a way that it exhibits a runtime behavior that is identical to the Central PDP's (which has originally been implemented in Java). The Change Tracker and the Local PDP leverage the Web Storage API[3]. More specifically, the `window.localStorage` JavaScript object provided by this API is used to permanently persist tracked model changes, the required entailment constraint model as well as the Local Model in the user's browser. The Change Tracker uses the `window.navigator.onLine` attribute as well as the emitted `online` event to react to regained network connectivity. The Application Cache[4] feature of HTML5 permanently caches all required assets (e.g., JavaScript or CSS files) upon accessing the Web application for the first time. This allows it to be accessed and executed in offline mode.

All server-side components are implemented in Java. More specifically, the Merge Service and the Access Control Model Repository are implemented as plain HTTP Services using the JAX-RS API. Internally we work with Ecore[5] model instances that are marshalled into JSON for the client-side application code. We use a model-driven approach for defining forms and documents and securing them using entailment constraints.

5. EVALUATION

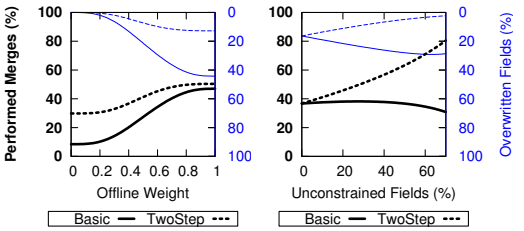
In this section we first evaluate the merge algorithms' possible degree of automation as well as the runtime performance and scalability of a prototype implementation. Then, we evaluate the performance and scalability of our prototype Merge Service. Finally, we contrast two implementation variants of our proposed snapshotting approach.

5.1 Evaluation of the Merge Algorithms

A crucial part of our approach both in terms of the possible degree of automation and the overall performance are the merge algorithms, which we evaluate in this section.

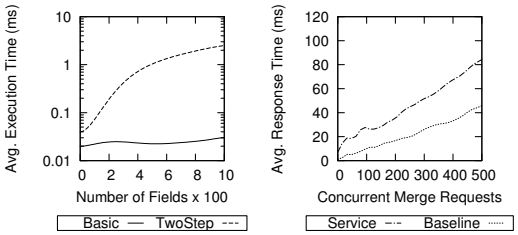
We evaluate the merge algorithms' possible degree of automation by analyzing and comparing the percentage of actually *performed merges* and of *overwritten fields*. The evaluations have been performed using an exemplary, typical

[1] `http://sharejs.org`
[2] `http://opencoweb.org`
[3] `http://www.w3.org/TR/webstorage`
[4] `http://www.w3.org/TR/html5/browsers.html#offline`
[5] `http://www.eclipse.org/modeling/emf`



(a) Varied Offline Weights (b) Constrained vs. Unconstrained Fields

Figure 5: Evaluation of Two Merge Algorithms



(a) Algorithms (b) Merge Service

Figure 6: Performance Evaluations

document and entailment constraint model with two subjects, one role and four data fields. Three of the data fields are constrained by subject-binding constraints and the remaining field is unconstrained. For this model, we have calculated all permutations of possible merge combinations. For these combinations, first, we contrast the basic (i.e., Algorithm 1) and two-step merge algorithms (i.e., Algorithm 3), the actually performed merges, and overwritten fields for different values of the offline weight parameter. Second, we contrast the performed merges and overwritten fields for different portions of unconstrained fields to analyze the impact of changes in the document and entailment constraint model.

Our approach introduces the notion of an offline weight as a means of discriminating offline changes (see Section 3.3). Figure 5a depicts how the offline weight affects the automatability, i.e., the chance that a merge can be performed automatically. For instance, in our exemplary scenario and an offline weight of 0.4 nearly 16.5% of all possible merge combinations could be merged using the basic merge algorithm, while 33% could be merged using the two-step merge algorithm.

On the other hand, a growing offline weight value leads for both algorithms to more deleted and eventually overwritten fields of the Shared Model. Figure 5a provides evidence that – at least in our exemplary scenario – the two-step merge algorithm always performs better than the basic merge algorithm, both in terms of a higher number of performed merges and a lower number of overwritten fields.

While the exact progression of the graphs is specific to our exemplary model, similar tradeoff graphs can be calculated for other models. That is, on the one hand, a higher offline weight increases the automatability, and, on the other hand, it also increases the number of situations where online data fields have to be overwritten. Consequently, there is no universally optimal offline weight, but it has to be determined empirically for a specific document and entailment constraint model.

To illustrate and analyze the impact of changes in the document and entailment constraint model, we set the offline weight parameter to 0.5 and change the number of unconstrained fields in the document. In Figure 5b we can see the effects of changing the ratio of constrained vs. unconstrained fields in our exemplary scenario. Obviously, for a model that has no constrained fields at all, the results are identical for both algorithms. If half of all fields in our exemplary model are unconstrained, the two-step merge algorithm manages to merge nearly twice (i.e., 62%) as often and exhibits nearly a seven times lower chance of overwritten fields compared to the basic merge algorithm. The gap between the two algorithms gets even wider with a growing ratio of unconstrained fields. In summary, we observed that in both scenarios (i.e., with changing offline weights and especially with a growing number of unconstrained fields in a model) the two-step merge algorithm exhibits significantly better characteristics than the basic merge algorithm.

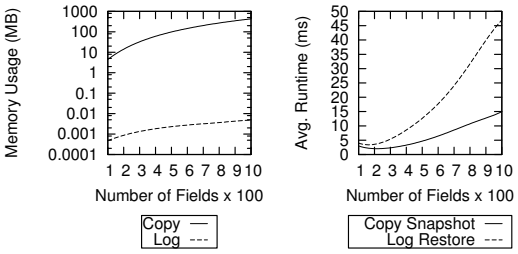
Algorithm 2 has a worst-case runtime complexity of $O(N \times M)$, i.e., in simplified form we can write $O(N^2)$. As a consequence, both merge algorithms (i.e., Algorithm 1 and 3) also have a runtime complexity of $O(N \times M)$.

Because of the quadratic complexity and because Algorithm 3 invokes Algorithm 1 two times, we further studied the performance impact of both approaches. The presented measurements are based on our prototype implementation (see Section 4) and have been conducted on a machine equipped with a 2.4 GHz dual core CPU, 8 GB RAM, running Ubuntu GNU/Linux 13.04. The performance is mainly dependent on the number of fields in a document. Figure 6a visualizes the average execution time of a merge operation for various field counts. Note that the y-axis is scaled logarithmically. Although the measurements exhibit a significantly higher average execution time for the two-step merge algorithm, the performance penalty should be negligible for "reasonable" field counts. We consider the execution times for both approaches to be acceptable, especially when keeping in mind that merges typically happen only occasionally.

5.2 Evaluation of the Merge Service's Performance and Scalability

In addition to the performance evaluations for the merge algorithms, we analyzed the performance of our prototype Merge Service. In particular, we analyzed how well our Merge Service handles merge requests that are issued simultaneously by a potentially large number of users.

Figure 6b compares the average response times of the Merge Service (using the two-step merge algorithm) and a "Baseline Service" (i.e., no computation at all), for a given number of simultaneous requests. For instance, in the case of 250 requests, the average response time for all clients is roughly 40ms and 20ms for the Baseline Service. This means, that in this case it takes roughly 20ms to perform the actual merge operation, while the remaining 20ms account for the underlying communication and Web Service stack. Our results indicate linear scalability and even in



(a) Memory Consumption (b) Execution Time

Figure 7: Evaluation of Snapshotting Approaches

the case of 500 users simultaneously submitting their offline changes, the average response time remains below a tenth of a second. As the Baseline Service represents the theoretical minimum that is possible for the given Web Service framework, we consider the performance overhead of our Merge Service to be acceptable in most scenarios.

Note that we used the same machine and environment as for the measurements of the merge algorithms. Additionally, both the services and the testing tool, i.e., Apache's `ab` tool[6] ran on the same machine. Hence, the measurements are free from network-induced effects such as latency or jitter.

5.3 Evaluation of Snapshotting Approaches

In Section 3.2 we motivated the need for a versionized Shared Model and proposed a snapshot-and-replay approach. A document with n data fields potentially requires n snapshots (i.e., if the fields are filled out one after the other). Hence, we measured the consumption of computing resources with a growing n. More specifically, we contrast two implementation variants. First, the *Copy* approach stores a clone of the complete document for each snapshot. Second, the *Log* approach merely stores a log entry (i.e., who changed which fields). Figure 7a illustrates the amount of memory that is required to hold n snapshots of a document of n data fields in memory for both approaches using our prototype implementation. Obviously, the *Copy* approach requires approximately four orders of magnitude more memory than the *Log* approach.

The corresponding (average) time that is needed to create a single snapshot (i.e., to clone a document) using the *Copy* approach is depicted in Figure 7b. As creating a snapshot using the *Log* approach has a runtime complexity of $O(1)$ we can neglect it. On the other hand, using the *Log* approach it takes significantly longer to restore a snapshot (i.e., the Shared Model has to be cloned and all changes that have been performed after the snapshot version to-be-restored have to be reverted in the cloned model) than for the *Copy* approach (which is $O(1)$). In summary, we conclude that the *Log* approach is considerably more resource efficient than the *Copy* approach.

6. RELATED WORK

In this section we discuss existing work in the related areas of Web collaboration platforms, access control enforcement, and (Web) document consistency.

[6] `http://httpd.apache.org/docs/2.4/programs/ab.html`

Web Collaboration Platforms. Systematic development of Web collaboration platforms has received considerable attention. Heinrich et al. [13] propose a collaboration infrastructure aimed at transforming single-user Web applications into collaborative multi-user applications by synchronizing DOM trees. Farwick et al. [8] discuss an architecture for Web-based collaborative meta-modeling. Their framework allows multiple users to work on graphical meta-models collaboratively. Modifications of the (meta-)models are secured by basic access control measures, but in contrast to our work, they do not explicitly address dynamic updates resulting from merging offline actions under RBAC entailment constraints. Our work builds on frameworks and libraries that facilitate the development of collaborative Web applications. For instance, the Open Cooperative Web Framework (see Section 4) consists of a set of JavaScript libraries and a generic Java servlet. MobWrite[7] is another approach for enabling real-time collaboration. However, it is restricted to synchronizing HTML forms, and the re-usability and applicability is thus somewhat limited.

Security and Access Control Enforcement. Throughout the last years, a plethora of approaches have been presented for integrating security and access control in Web applications. Joshi et al. [15] provide an early study on generic security models for Web-based applications. In [4], Belchior et al. model RBAC policies using RDF triples and N3Logic rules. Tackling security on the client side, Guarnieri et al. [12] propose the GATEKEEPER framework for authoring and enforcing policies in JavaScript code. This complements our approach of having a local PDP component on the client devices. Ahn et al. [1] present an approach for injecting RBAC into an already existing Web-based workflow system. They propose a special reverse proxy for enforcing RBAC rules transparently to the actual Web application. Sohr et al. [19] and Hummer et al. [14] propose a similar approach in the context of Web services, using interceptors that are able to prevent the actual invocation in case of a policy violation. Several works on distributed security enforcement, particularly in mobile networks, complement and have influenced our approach. For instance, Alicherry et al. [2] support offline nodes by early distribution of policy tokens among network nodes. Similarly, Gasmi et al. [9] partition networks into trusted virtual domains which can operate and manage access rights independently. Finally, while we focus on access control, various other issues and threats also need to be taken into account, as studied by Almorsy et al. [3]. Their approach utilizes Object Constraint Language to define vulnerability signatures for threats like SQL injections or cross site scripting. Based on the signatures, different mitigation actions are proposed.

Document Consistency. The seminal work of Sun et al. [21] proposes the transparent adaptation (TA) approach to develop collaborative multi-user applications. The cornerstone of TA is operational transformation (OT) [7]. Given two concurrent operations o_1 and o_2 resulting in states s_1 and s_2, the core idea of OT is to transform the parameters of the operations to execute them on the current state while maintaining document consistency. Our approach is orthogonal to OT: the RBAC policies and entailment constraints

[7] `https://code.google.com/p/google-mobwrite`

provide an application workflow with well defined responsibilities, and we maintain document consistency by allowing only sequences of operations that comply with this workflow. The problem of consistency with concurrent modifications has also been a core issue in database research. The general approach of allowing concurrent work, followed by merging and resolving conflicts is denoted *optimistic concurrency control* [5] in databases. In this sense, the sequence of offline actions in our approach relates to a transaction in databases, and during merging the action sequences "compete" with the online changes that have happened in the meantime. A consistent merging strategy that could be applied to this problem is discussed and formally verified by Lin and Mendelzon in [16].

7. CONCLUSION AND FUTURE WORK

In this paper we have shown that offline editing for entailment constrained, real-time collaborative Web documents can effectively be realized using a combination of client-side access control enforcement and a document merging approach. We highlighted that merging such documents is inherently prone to conflicts and motivated the need for a merge approach that is capable of detecting and resolving conflicts automatically. We provided evidence that many possible conflicts can be resolved automatically and that both the merge algorithms and the prototype Merge Service work with acceptable runtime performance and scalability even for lots of simultaneous merge requests and documents with lots of data fields. In addition, we demonstrated that modern Web browsers as well as HTML5 and some of its accompanying APIs provide a platform that can be used to implement our novel approach.

Future work will address limitations inherited from HTML5 and Web browser implementations, such as the limited client-side storage capacity which be problematic if we have to deal with huge document and entailment constraint models (i.e., tens of thousands of model elements). Although we have shown that our basic merge approach works quite well, the characteristics of the two-step merge approach leads us to believe that there is still room left for improvements in that area. Another interesting topic would be to devise an approach for estimating (i.e., instead of determining it empirically) the optimal offline weight for a given document and the corresponding entailment constraint model. Furthermore, we will apply our approach to other types of collaborative processes. In particular with regard to dynamic processes (e.g., free text editing or modeling) we will have to deal with completely dynamic document and access control and constraint models (i.e., models that change at runtime).

8. REFERENCES

[1] G.-J. Ahn, R. Sandhu, M. Kang, and J. Park. Injecting RBAC to secure a Web-based workflow system. In *5th ACM workshop on RBAC*, pages 1–10, 2000.

[2] M. Alicherry, A. Keromytis, and A. Stavrou. Deny-by-default distributed security policy enforcement in mobile ad hoc networks. In *5th SecureComm*, 2009.

[3] M. Almorsy, J. Grundy, and A. S. Ibrahim. VAM-aaS: Online Cloud Services Security Vulnerability Analysis and Mitigation-as-a-Service. In *13th WISE*, 2012.

[4] M. Belchior, D. Schwabe, and F. Silva Parreiras. Role-based access control for model-driven web applications. In *12th ICWE*, pages 106–120, 2012.

[5] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[6] E. Bertino, E. Ferraria, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *TISSEC*, 2(1), 1999.

[7] C. Ellis and S. Gibbs. Concurrency control in groupware systems. *SIGMOD Record*, 18(2):399–407, 1989.

[8] M. Farwick, B. Agreiter, J. White, et al. A web-based collaborative metamodeling environment with secure remote model access. In *10th ICWE*, 2010.

[9] Y. Gasmi, A.-R. Sadeghi, et al. Flexible and secure enterprise rights management based on trusted virtual domains. In *3rd ACM STC workshop*, 2008.

[10] P. Gaubatz, W. Hummer, U. Zdun, and M. Strembeck. Supporting customized views for enforcing access control constraints in real-time collaborative web applications. In *13th ICWE*, 2013.

[11] P. Gaubatz and U. Zdun. Supporting entailment constraints in the context of collaborative web applications. In *28th Symposium On Applied Computing*, 2013.

[12] S. Guarnieri and B. Livshits. GATEKEEPER: mostly static enforcement of security and reliability policies for javascript code. In *USENIX Security'09*, 2009.

[13] M. Heinrich, F. Lehmann, T. Springer, and M. Gaedke. Exploiting single-user web applications for shared editing: a generic transformation approach. In *21st Int. Conf. on WWW*, pages 1057–1066, 2012.

[14] W. Hummer, P. Gaubatz, M. Strembeck, U. Zdun, and S. Dustdar. An integrated approach for identity and access management in a SOA context. In *16th ACM SACMAT Symposium*, pages 21–30, 2011.

[15] J. Joshi, W. Aref, A. Ghafoor, and E. Spafford. Security models for web-based applications. *Communications of the ACM*, 44(2):38–44, 2001.

[16] J. Lin and A. O. Mendelzon. Merging databases under constraints. *Int. J. of Coop. Inf. Systems*, 07(01), 1998.

[17] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *Computer*, 1996.

[18] S. Schefer, M. Strembeck, J. Mendling, and A. Baumgrass. Detecting and resolving conflicts of mutual-exclusion and binding constraints in a business process context. In *19th CoopIS*, 2011.

[19] K. Sohr, T. Mustafa, X. Bao, and G.-J. Ahn. Enforcing Role-Based Access Control Policies in Web Services with UML and OCL. In *24th ACSAC*, 2008.

[20] M. Strembeck and J. Mendling. Generic algorithms for consistency checking of mutual-exclusion and binding constraints in a business process context. In *18th CoopIS*, pages 204–221, 2010.

[21] C. Sun, S. Xia, et al. Transparent adaptation of single-user applications for multi-user real-time collaboration. *ACM TOCHI*, 13(4):531–582, 2006.

[22] J. Wainer, P. Barthelmes, and A. Kumar. W-RBAC - A workflow security model incorporating controlled overriding of constraints. *Coop. Inf. Syst.*, 12(4), 2003.

Paper H

Two Controlled Experiments on Model-based Architectural Decision Making

The subsequent paper has been submitted as follows:

I. Lytra, P. Gaubatz, and U. Zdun. Two Controlled Experiments on Model-based Architectural Decision Making. submitted to *Information and Software Technology*, submitted first revision in January 2015.

In this paper we presented our third research prototype implementation, called CoCoADvISE (Constrainable Collaborative Architectural Design Decision Support Framework). CoCoADvISE is a real-time collaborative Web application that provides means for making and documenting architectural design decisions in a collaborative way. Such architectural design decisions have been becoming more and more common for documenting software architectures, in recent years. Recent studies (see, e.g., [65, 74, 75]) also showed that such decisions are usually made in a collaborative manner. However, there has been little empirical evidence about the supportive effect of reusable architectural knowledge on the effectiveness and efficiency of architectural decision making. To investigate these aspects, we conducted two separate controlled experiments with novice architects in which we tested the supportive effect of reusable decision models in decision making and documentation. We could show that the CoCoADvISE approach significantly increased both the efficiency and the effectiveness of novice architects.

Two Controlled Experiments on Model-based Architectural Decision Making

Ioanna Lytra*, Patrick Gaubatz, Uwe Zdun

Software Architecture Research Group, University of Vienna, Austria

Abstract

Context: In recent years, architectural design decisions are becoming more and more common for documenting software architectures. Rather than describing the structure of software systems, architectural decisions capture the design rationale and – often reusable – architectural knowledge. Many approaches and tools have been proposed in the literature to support architectural decision making and documentation (for instance, based on models, ontologies, or templates). In this context, the capturing, organization, and effective reuse of architectural knowledge has gained a lot of attention. **Objective:** However, there is little empirical evidence about the supportive effect of reusable architectural knowledge on the effectiveness and efficiency of architectural decision making. **Method:** To investigate these aspects, we conducted two separate controlled experiments with software architecture students in which we tested the supportive effect of reusable decision models in decision making and documentation. **Results:** Our results show that the use of reusable decision models can significantly increase both the efficiency and the effectiveness of novice architects. **Conclusion:** We can report, that our findings are in line with similar studies and support the claims regarding reusable architectural design decisions in principle.

Keywords: architectural design decision, architectural decision model, architectural knowledge, controlled experiment

1. Introduction

In recent years, architectural design decisions (ADDs) have been promoted to first class citizens in software architecture documentations [1]. Rather than documenting the structure of software systems (e.g., components and connectors), ADDs contribute to the capturing of design rationale. There are numerous attempts on documentation and leveraging of design rationales with focus on

*Corresponding author: Ioanna Lytra, Faculty of Computer Science, University of Vienna, Währingerstraße 29, 1090 Vienna, Austria; Phone, +43-1-4277-78523

Email addresses: ioanna.lytra@univie.ac.at (Ioanna Lytra), patrick.gaubatz@univie.ac.at (Patrick Gaubatz), uwe.zdun@univie.ac.at (Uwe Zdun)

the reduction of architectural knowledge (AK) vaporization [2], reusability of ADDs [3], and AK sharing [4]. Apart from that, the documentation of ADDs for providing architectural guidance in software projects has gained much attention in industrial practice [5, 6], lately. In this context, capturing the design solutions and their rationale is important not only for the experienced software architects but also for novice software designers who need to be educated on the existing AK and the systematic reasoning on ADDs to avoid both reinventing the wheel and making ADDs of bad quality.

Reusing ADDs can contribute to simplifying architecting [7]. Thus, addressing systematic documentation of ADDs and providing guidance during decision making for recurring design issues, the use of reusable ADD models has been proposed in the literature [3]. Similar to leveraging patterns for architectural decision making [2], reusable ADD models provide proven solutions – both application generic and technology specific – to various design issues along with their forces and consequences. Examples of reusable ADD models that have been documented cover solutions for designing service-oriented architectures (SOA) [8] and service-based platform integration solutions [9].

A few reusable ADD models and related tools that support their management (such as [10]) have been evaluated in real-life contexts. For instance, Zimmermann et al.'s ADD model consisting of 300 ADDs from the SOA domain covering various aspects such as Web service integration and transaction management has been used by practitioner communities and in industrial projects [8]. However, no feedback or empirical evidence has been gathered on whether and to which extent reusable ADD models are beneficial (i.e., they support effectiveness and efficiency of architects) in the architectural decision making process. While a few studies have investigated how reusable AK management and sharing is practiced in industrial contexts [5, 11] and have validated the supportive effect of pattern-based reusable AK in the decision making process [12], the software architecture community lacks empirical evidence on the positive impact of reusable AK on ADD making and documentation. Such empirically-grounded findings are important not only for validating the benefits of reusable AK in practice, but also for understanding, improving, and supporting the management and leveraging of reusable ADDs.

Therefore, we conducted two controlled experiments with students to test whether the use of reusable ADD models increases the efficiency and effectiveness of architects in the decision making and documentation process. We explicitly considered software architecture students in our evaluation, as reusable ADD models are supposed to be used as guidance models by trainers for systematically teaching patterns and technology best practices to new or inexperienced members in a software development team [13]. In the two controlled experiments, 49 and 122 students, respectively, with background in software architecture and design patterns, were asked to make and document ADDs while designing the architecture of two different software systems. For this, a Web-

based tool support, called CoCoADvISE[1], was provided to the experiment and control groups. Both the experiment and control group received material with related patterns and technology documentations and could use the tool to make and document decisions. Contrary to the control group, the tool provided additional semi-automated decision making guidance based on reusable ADD models for the participants of the experiment group. We found that participants who were supported by our semi-automated decision making guidance approach ...

- delivered more documented ADDs.
- delivered ADDs of better quality.
- invested less time for documenting ADDs.

The remainder of the paper is structured as follows. We give an overview of the approaches related to architectural decision making and documentation and compare existing architectural decision management tools with CoCoADvISE in Section 2. We present our Web-based tool CoCoADvISE for decision making and documentation and discuss its main concepts in Section 3. In Sections 4 and 5 we describe our experimental settings, as well as the analysis of the results for the two controlled experiments we conducted. Our findings, implications, and validity threats are discussed in Section 6, and finally, Section 7 concludes with a summary of our contributions and discusses future work.

2. Related Work

In this section, we discuss the concept of ADDs, present existing tools and methods for decision making and documentation, and summarize the few empirical studies related to ADDs that exist in the literature.

2.1. Architectural Design Decisions

ADD documentations contain not only the resulting designs but also the justification, the strengths and weaknesses, as well as alternatives for the selected design solutions. Thus, software architects capture ADDs for analyzing and understanding, as well as sharing and communicating the rationale and implications of these decisions. Apart from that, the documentation of ADDs prevents the potential loss of AK, a phenomenon which is known as *architectural knowledge vaporization* [1, 2]. There are numerous attempts on supporting ADDs and capturing their design rationales. Clements et al. suggest a general outline for documenting architectures and guidelines for justifying design decisions [15] while Tyree and Akerman present a rich template for capturing

[1] CoCoADvISE is the web-based version of our previous tool ADvISE for decision making and documentation [14] and shares common concepts with other reusable ADD approaches that have been documented in the literature (such as [3]). CoCoADvISE was developed for the needs of the controlled experiments and in order to provide additionally collaboration support.

and documenting several aspects of ADDs [16]. A different technique proposed by Lee and Kruchten aims at establishing formalized ontological descriptions of architectural decisions and their relationships [17]. Zimmermann et al. use decision meta-models to capture reusable AK [3] to be reused among different projects of the same domain. In addition, patterns are regarded proven knowledge for capturing ADDs and their rationale [2] and are considered often in the aforementioned approaches.

Numerous tools for the management of ADDs have been proposed in the literature [18–20]. In addition, a substantial amount of work has been done in the direction of documenting the AK using architectural decision modeling (refer to [18] for a comparison of existing architectural decision models and tools). For instance, Jansen and Bosch propose a meta-model for capturing decisions that consist of problems, solutions and attributes of the AK [1]. Zimmermann et al.'s meta-model for capturing ADDs [6] consists of three core domain entities: *Architectural Decision (AD)* related to one or more *ADTopics* organized in *ADLevels*, entailing *ADAlternatives*, the selection of which leads to an *ADOutcome*. The advantage of such ADD models is that they are reusable and can be used as guidance for architectural decision making activities, whenever recurring design issues emerge. Reusable ADD models share common concepts with patterns (see [2]), that is, they both provide proven solutions for specific design issues along with their motivation and rationale. The main difference is that reusable ADD models provide the means for defining formally more complex relationships for ADDs (e.g., the selection of a design option may exclude a design solution). Furthermore, they allow us to capture except for generic knowledge – usually addressed by patterns – also domain and technology specific AK. Yet, the relationship between architectural patterns and reusable ADD models can be eventually synergetic [3], for instance, reusable decision models can be integrated with patterns and guide the selection of patterns. Various reusable ADD models have been documented in the literature, covering SOA-related solutions [8], service-based platform integration [9], the design of domain specific languages [21], and model and metadata repositories [22].

In our empirical study, we focus on the evaluation of reusable AK in the form of reusable ADD models. For this, we provide reusable ADD models for the participants of the experiment groups of the two controlled experiments similar to the aforementioned reusable ADD models.

2.2. Tools for Architectural Decision Making and Documentation

Several tools have been developed to ease capturing, managing and sharing of architectural decisions. In most of the cases, the focus is set on the manipulation of architectural decision artifacts and their relationships, and the capturing and reuse of AK, as well as collaboration aspects. In our work, we do not intend to develop "yet another tool" for ADD management but rather to implement existing concepts in architectural decision support such as reusable architectural decision models [3] and the Questions-Options-Criteria (QOC) approach [23] and provide semi-automated tool support integrating these concepts. Our main goal is to gather empirical evidence on the supportive effect of reusable ADDs

in architectural decision making. In this section, we discuss existing tools for architectural decision making and documentation and compare these to CoCoADvISE, the Web-based tool we have evaluated in our empirical study.

PAKME [24] and ADDSS [25] are some of the first attempts to develop tools for decision making and documentation. PAKME aims at providing AK management support for knowledge acquisition, maintenance, retrieval, and presentation, and contains features for capturing and classifying ADDs, integrating pattern-based knowledge, and relating ADDs with quality-based scenarios and requirements. Similar to PAKME, ADDSS can be used for storing, managing and documenting ADDs. CoCoADvISE targets both decision making and documentation. For the decision making, reusable ADD models are leveraged [3] and for the documentations, a text template like the one introduced by Tyree and Akerman [16] is used. Unlike tools that focus on the visualization and understandability of ADDs (such as [26, 27]), we mainly target decision guidance and automation of steps in decision making and documentation.

Many tools in the literature support the integration of ADDs in the software architecture and development processes. For instance, Archium aims primarily at capturing architectural decisions and weaving them into the development process, that is, binding them with models and system implementation [28]. Also, the ADVERT approach targets the capturing of pattern-specific ADDs with their rationale along with decision-relevant requirements, quality attributes, and architectural elements [29]. In our previous work, we have integrated ADDs modeled in CoCoADvISE tool with architectural views, in particular, component-and-connector views [14]. These aspects are out of the scope of this paper though.

Some of the tools have been developed with focus on collaboration between architects. Software Architecture Warehouse (SAW) supports collaborative architectural decision making and enhances situational awareness by providing an environment for real-time synchronization of design spaces, voting, and discussion for software architects [30]. The purpose of the wiki-based tool AD_{kwik} is also to support collaboration in decision making through sharing of knowledge about ADDs across project boundaries [31].

Although automated support is an important aspect in decision making, it is addressed very little by existing approaches. Ameller et al. propose Architech to support software architects during the architectural decision-making process by suggesting alternative decisions for satisfying non-functional requirements [32]. For this, optimization techniques are applied on an AK ontology. Also, ArchDesigner provides a quantitative quality-driven approach to find the best possible fit between various quality goals, competing architectural concerns and constraints [33]. It uses Multiple Attribute Decision Making (MADM) and optimization techniques to find a best-fitting architecture composed of interdependent architectural design decisions that satisfies prioritized quality goals. CoCoADvISE provides semi-automated support at the following steps: (1) decision making is guided through questionnaires and (2) semi-complete ADD documentations are generated based on suggested design solutions.

Similar to our tool, Zimmermann et al. provide an architectural design

method to combine pattern languages with reusable architectural decision models [6]. The goal of their approach is to provide domain-specific pattern selection advice and traceability links from platform-independent patterns to platform-specific decisions. However, the reusable ADD models have to be used manually by software architects while in CoCoADvISE automated decision guidance based on reusable decision models is provided.

The majority of the proposals have been evaluated either in case studies (e.g., [6, 26]), in industrial projects [25], or focus groups [30], and in a few cases no evaluation is reported. None of the tools has been empirically validated and only little feedback has been gathered by the users[2]. We conducted two controlled experiments with 171 software architecture students in total to test the supportive effect of the main concept of CoCoADvISE, namely the reusable ADD models, on capturing ADDs.

The reader can refer to Table 1 for an overview of the tools discussed in this subsection. In particular, we report on the type of the tool support (decision making or/and documentation), whether it provides automation support, and how it has been evaluated.

2.3. Other Empirical Studies Related to ADDs

There are various literature surveys and reviews on the approaches and tools for architectural decision making and documentation. For instance, Falessi et al. provide a comparison of various techniques for selecting architectural alternatives during decision making [7]. Shahin et al. provide a survey on existing ADD models and tools [18] while Bu et al. provide an analysis of decision-centric approaches for design support [36]. The mapping study by Tofan et al. provides an overview and taxonomy of the existing literature on ADDs [20]. Furthermore, Weinreich and Groher compare software architecture management approaches with focus on the main aims of documenting AK as well as the elements used to represent AK [37].

Except for these literature surveys and reviews, little empirical evidence (especially quantitative results) exists on the use and benefits of ADDs in the industry and by practitioners. Heesch et al. conducted a multiple-case study with four teams of software engineering students working in industrial projects in order to find out whether decision documentation supports novice architects in following a rational design process [38]. Shahin et al. test their hypothesis that the visualization of ADDs and their rationale improves the understanding of architecture designs in a controlled experiment with 10 participants [34]. Also, the supportive effect of pattern use in recovering of ADDs in terms of quality and quantity of documented ADDs has been validated in a controlled experiment with 33 software architecture experts and practitioners in a different study [12]. A few other qualitative studies such as [39, 40] focus on how software

[2]Except for two studies [30, 34] which contain only a very brief summary of the results and do not study their statistical significance.

Name	Decision Making	Documentation	Automation Support	Approach Evaluation
ArchPad [6]	+	−	−	Case study from the finance industry. SOA-related decisions (300) are documented.
ArchiTech [32]	+	−	Suggests alternative decisions based on NFRs	No evaluation reported
AD_{kwik} [31]	−	+	−	SOA-related decisions (300) are documented
Archium [28]	+	+	−	Motivating example
MAD 2.0 [26]	+	+	−	Case study (CRM system)
PAKME [24]	+	+	−	No evaluation reported
Compendium [27]	+	+	−	SOA-related decisions are documented
ADDSS [25]	+	+	−	Two industrial projects (multimedia system and virtual reality application)
ADVERT [29]	+	−	−	Evaluation using the Common Component Modeling Example (CoCoME)[1]
ArchDesigner [33]	+	−	Making trade-offs between stakeholders	Industrial project (Glass Box)
SAW [30]	+	−	−	In a focus group. 20 participants considered 100 issues with 5 alternatives each.
CoCoADvISE	+	+	Predefined questionnaires used during decision making	Two controlled experiments with 49 and 122 students respectively

[1] CoCoME is an example system which provides a benchmark for component-based modeling approaches [35].

Table 1: Comparison Overview of ADD Tools

architects make and document ADDs and which of them are being eventually documented. To the best of our knowledge, no empirical-grounded evidence exists yet on the impact of reusable AK and in particular reusable ADD models on the efficiency and effectiveness of software architects.

3. Architectural Decision Making Using CoCoADvISE

In this section, we introduce the CoCoADvISE prototype. In particular, we discuss its ADD meta-model (see Section 3.1) and core functionalities (see Section 3.2), and provide selected implementation details (see Section 3.3). CoCoADvISE (Constrainable Collaborative Architectural Design Decision Support Framework) provides semi-automated support for architectural decision making and documentation. In addition, it supports collaboration in decision making under various stakeholders' constraints, but this will not be discussed further in the current paper. The Web-based version of the tool was developed for the needs of the controlled experiments in such a way that our approach could be generalizable to a big extent. That is, the utilization of reusable ADD models, as well as the tasks – to be performed by the software architects – for making and documenting ADDs (see Section 4 for details) were designed in such a way that the obtained results can be generalizable and not bound to the use of the specific tool.

3.1. Reusable ADD Meta-model

CoCoADvISE provides tool support for modeling of reusable ADDs inspired by the Questions, Options, and Criteria (QOC) approach [23]. The CoCoADvISE ADD model extends the traditional QOC approach with – among others – additional dependencies between options, such as enforcement or inclusion, options and questions or decisions (i.e., an option triggers a next question or decision). In addition, it supports categorizing reusable solutions (i.e., options in QOC).

CoCoADvISE introduces a reusable ADD meta-model (see Figure 1) for the design space of certain application domains, consisting of *Decisions*, *Questions*, *Options*, and *Solutions*, as well as various relationships among them. For each design issue, a set of *Questions* (including one or more first *Questions*) providing multiple *Options* has to be modeled. Examples of relationships are that a selection of an *Option* triggers a follow-up *Decision* or an *Option* is incompatible with or enforces another *Option*. A selection of an *Option* may lead to a *Solution* to be suggested to the software architect. This *Solution* will be applied using an *Architecture Guidance*, that is a *Technology-related AK*, a *Design Pattern*, an *Architectural Pattern*, or a *Composite Reusable Solution* combining any of the aforementioned Architectural Guidance types.

The concepts that we introduce in CoCoADvISE are comparable to existing reusable ADD approaches such as the ones documented in [3, 9, 22]. Therefore, our hypotheses testing and evaluation results are considered to apply on similar approaches as well.

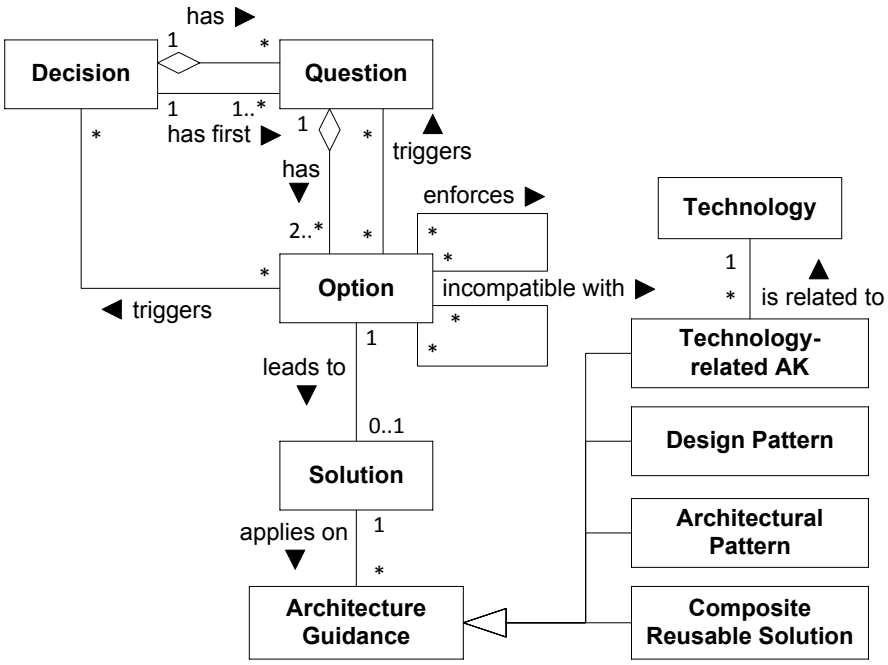


Figure 1: Reusable ADD Meta-model

3.2. Decision Support Based on Reusable ADD Models

The advantage of CoCoADvISE's reusable ADD models is that they are created only once for a recurring design situation and can be reused multiple times following the model-driven paradigm. In similar application contexts, corresponding questionnaires can be generated and used for making concrete decisions. Based on the outcomes of the questionnaires answered by software architects through the decision making process, CoCoADvISE can automatically resolve potential constraints and dependencies (e.g., reveal follow-on questions and decisions, deactivate incompatible options, etc.), recommend best-fitting design solutions, and eventually, generate half-filled ADD documentations. Using CoCoADvISE the software architect can choose from a list of reusable ADD models (see ① of Figure 2) and generate a questionnaire based on this model. The questionnaire provides architectural decision guidance through single-choice questions which lead to follow-on questions, decisions, and recommendations as indicated in ②. Finally, software architects may generate semi-complete documentations based on the CoCoADvISE recommendations such as the one shown in ③. CoCoADvISE fills in automatically some of the fields of the ADD documentations in template form [16]; architects need to fill in afterwards the rest of the required information.

3.3. Prototype Implementation Details

CoCoADvISE is the Web-based version of our previous Eclipse-based tool ADvISE [14] and was developed for the needs of the controlled experiments.

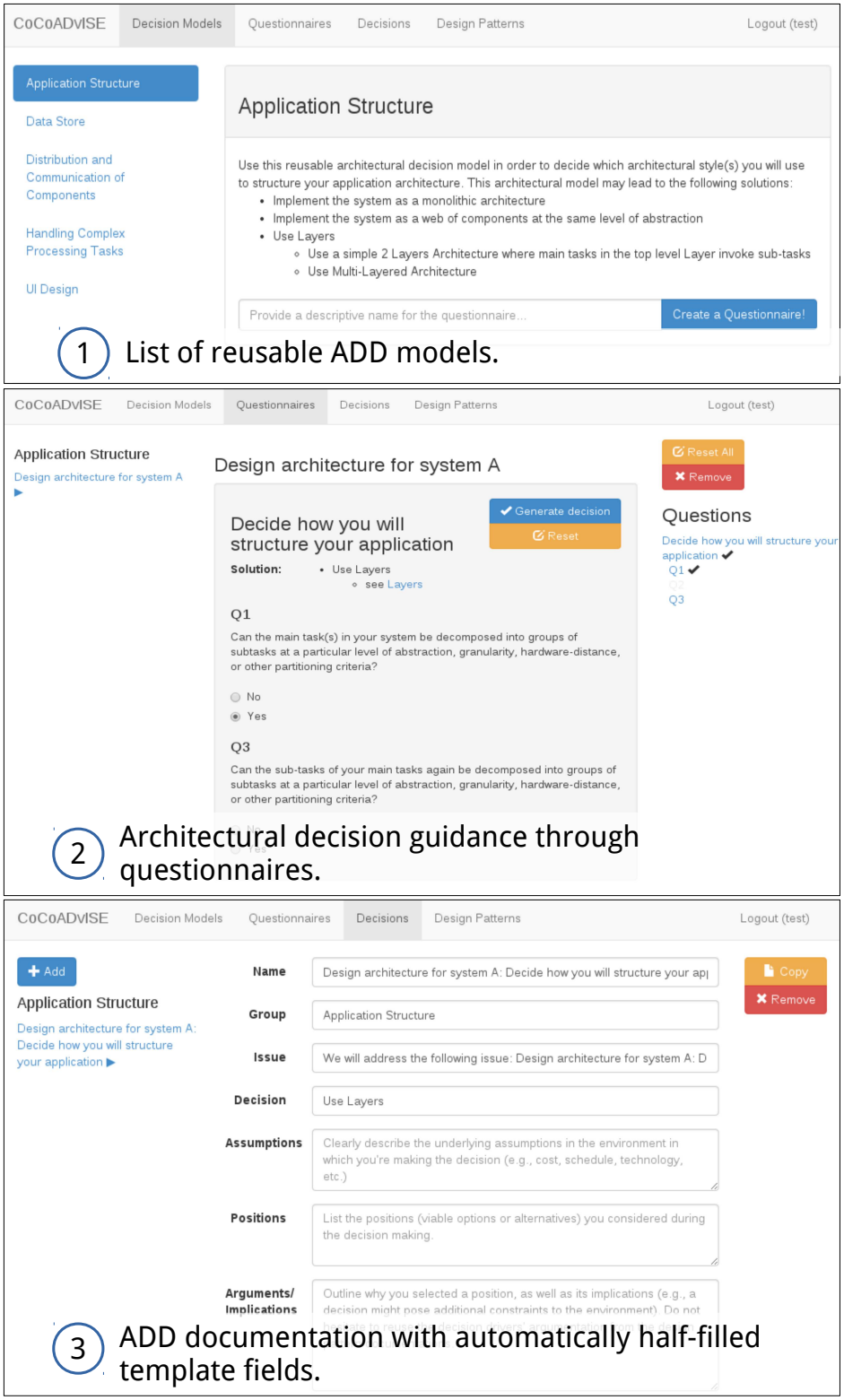


Figure 2: Screenshots of CoCoADvISE

It supports additionally collaborative architectural decision making. CoCoADvISE is a mostly client-side executed Single-page Web application that is implemented using Google's AngularJS[3] framework. The back-end of this Thin Server Architecture is founded on the Node.js[4] framework and runtime environment. In particular, it utilizes the real-time synchronization engine Racer[5] and persists the application state in a MongoDB[6] database.

4. Controlled Experiments

To measure the effect of using reusable architectural decision models on the efficiency and effectiveness of software architecture students we have conducted two separate controlled experiments with software engineering students. For the design and execution of the two experiments, we followed the guidelines of Kitchenham [41] while for the analysis, evaluation, and presentation of results we have consulted the advice of Wohlin et al. [42]. In the following subsections, we discuss the common goals and hypotheses of the controlled experiments and present their design and execution in detail.

4.1. Goals and Hypotheses

The goal of the experiments is to study and quantify the benefits of using CoCoADvISE and in consequence reusable ADD models for making and documenting ADDs in terms of quantity and quality related effectiveness, as well as time efficiency. For this, we compare two groups of students, one using reusable ADD models and one using an ad-hoc process based on pattern documentations to make and document ADDs in template form.

We postulate the following three null hypotheses and corresponding alternative hypotheses: Making and documenting ADDs using reusable ADD models...

H_{01} leads to lower or equal *quantity related effectiveness* of software architecture students compared to an ad-hoc process for architectural decision making.

H_1 leads to increased *quantity related effectiveness* of software architecture students compared to an ad-hoc process for architectural decision making.

H_{02} leads to lower or equal *quality related effectiveness* of software architecture students compared to an ad-hoc process for architectural decision making.

H_2 leads to increased *quality related effectiveness* of software architecture students compared to an ad-hoc process for architectural decision making.

H_{03} leads to lower or equal *time related efficiency* of software architecture students compared to an ad-hoc process for architectural decision making.

[3] `http://angularjs.org`
[4] `http://nodejs.org`
[5] `http://github.com/codeparty/racer`
[6] `http://mongodb.org`

H_3 leads to increased *time related efficiency* of software architecture students compared to an ad-hoc process for architectural decision making.

We perform statistical tests to find out whether the corresponding null hypotheses can be rejected. We expect that the users of CoCoADvISE will deliver ADDs of better quality and will manage to document more ADDs and in less time.

4.2. Parameters and Values

Several variables have been observed during the two experiments. In the following subsections, we discuss all dependent, independent, and derived variables we used for testing our hypotheses. A detailed list of variables (description, type, scale type, unit, and range) is provided in Table 2.

4.2.1. Dependent Variables

All dependent variables have been captured by and extracted automatically from CoCoADvISE's database. In particular, we instrumented its source code in such a way that we could precisely record all user activities within the system. The variable *time* indicates a single user's total time spent logged in in the application. The number of decisions that were documented by each student is indicated with the variable $numOfDecisions$. Finally, the variable *quality* refers to the quality of each ADD that was evaluated in a scale from 1 to 10 by two independent experts.

4.2.2. Derived Variables

To allow for a meaningful comparison of the *time* spent by the students to document the decisions we decided to introduce the variable $timeNorm$ which expresses the time needed per decision. In this way, we exclude the possibility that one treatment group needed less time to perform the tasks because they just delivered less work. In addition, we calculate the average points per student for the total of documented decisions ($qualityPerStudent$).

4.2.3. Independent Variables

The independent variables *group*, *exp* and *commExp* can potentially influence the dependent variables. In particular, *group* indicates the participant's treatment group (i.e., control or experiment), and *exp* and *commExp* refer to their programming experience in general and in the industry, respectively.

4.3. Experiment Design

The controlled experiments were conducted in the context of Information System Technologies and Software Architecture lectures respectively at the Faculty of Computer Science, University of Vienna, Austria. The first controlled experiment took place in Winter Semester 2013/2014 (January 2014) and the second in Summer Semester 2014 (June 2014). All participants were at the final semesters of their bachelor studies and had background in software architecture and design patterns. The readers can refer to Table 3 for a summary of

Type	Name	Description	Scale Type	Unit	Range
Dependent	*time*	Overall time needed to make and document decisions	Ratio	Minutes	Natural numbers including 0
	numOfDecisions	Number of documented decisions	Ratio	–	Natural numbers including 0
	quality	Quality of documented decision	Interval	–	1 (lowest) to 10 (highest)
Derived	*timeNorm* $(= \frac{time}{numOfDecisions})$	Time needed per decision	Ratio	Minutes	Natural numbers including 0
	qualityPerStudent $(= \frac{\sum_{i=1}^{numOfDecisions} quality}{numOfDecisions})$	Average quality of decisions	Interval	–	1 (lowest) to 10 (highest)
Independent	*group*	Treatment group	Nominal	–	Either "experiment" or "control"
	exp	Programming experience	Ordinal	Years	4 classes: 0-1, 1-3, 3-6, >6
	commExp	Commercial programming experience in industry	Ordinal	Years	4 classes: 0-1, 1-3, 3-6, >6

Table 2: Observed and Derived Variables

the experiment design in the two cases. We will refer to the first and second controlled experiment as *ORExp* (Online Retailer Experiment) and *UniISExp* (University Information System Experiment) respectively from the corresponding case studies that were used in each case.

	ORExp	**UniISExp**
Location	University of Vienna, Austria	University of Vienna, Austria
Time	January 2014	June 2014
Case Study	Online Retailer	University Information System
# Participants	49 (27 control / 22 exp.)	122 (56 control / 66 exp.)
# ADDs	319	762
# ADD Models	5 (16 patterns)	5 (40 patterns)

Table 3: Experiment Design Data

Participants. In the first controlled experiment in January 2014, from the 49 students of the lecture, 27 participated in the control group and 22 in the experiment group. In the second experiment in June 2014, a bigger sample of 122 students (56 in the control group and 66 in the experiment group) participated.

The experiments have been conducted in the course of mandatory practical exercises on architectural decisions for service-based software systems and distributed software systems respectively. The experiments took place in separate exercise groups (4 in the first and 6 in the second experiment) and in different computer labs at different times, which also explains the unequal number of participants in the corresponding control and experiment groups. The students in each group were assigned to the treatment groups randomly. In summary, 319 ADDs were documented in ORExp and 762 in UniISExp. All participants had experience with Java programming and design patterns and were familiar with the concepts of software architecture and architectural design decisions, as these topics had been already discussed in the corresponding lectures or were prerequisites to accomplish the previous exercises of the corresponding subjects.

Objects. A list of documented architectural design patterns and technology-related solutions was integrated in the CoCoADvISE tool for both groups in wiki format. The design patterns and architectural decision models were selected based on the lecture materials known to the students as well as the students' experiences from the previous programming exercises. The experiment group was provided additionally with a set of reusable architectural decision models and instructions how to use them for getting decision support with the aid of the CoCoADvISE tool.

Instrumentation. In the preparation phase, all students were asked to study the catalog of architectural design patterns and related technologies (afterwards integrated in the CoCoADvISE tool). Before starting with the experiment tasks,

all participants had to fill in a short questionnaire regarding their programming experience. Afterwards, the participants of both the control and experiment groups were provided with a description and requirements of the system to be designed. That was "An Online Retailer for Selling Books and Gadgets" and "A University Information System (IS)" for the ORExp and UniISExp experiments respectively. Both systems and their requirements were based on the experience of the authors with similar software systems from industrial projects. In Table 4 and Table 5 we give examples of the aforementioned software system's requirements. The complete descriptions of the design exercises can be found at `http://andromeda.swa.univie.ac.at/advise-experiment/exercises`.

Name	**Description**
Take Orders	Customers can place orders via two different channels: Web site and call center. Each of these systems is based on a different technology and stores incoming orders in a different data format. The call center system is a packaged application, while the Web site is a custom J2EE application. We want to treat all orders equally, regardless of their source. For example, a customer should be able to place an order via the call center and check the order status on the Web site. *Decide* how to deal with the different channels and feed the retailer with unified messages.

Table 4: Excerpt from the "Online Retailer for Selling Books and Gadgets" Requirements

Name	**Description**
Research Network	The University is collaborating with a central research system for establishing networks between researchers, dissemination of research results, access to publications, etc. The University IS should be able to send updates of new publications and research projects of its research staff. In this case, reliability and performance are not very important. The Research Network uses only the XML-RPC protocol for communication with external systems. *Decide* how you will design the communication between the University IS and the Research Network.

Table 5: Excerpt from the "University Information System (IS)" Requirements

The students had to consider six sets of requirements in ORExp (*Take Orders*, *Process Orders*, *Logging*, *Track Orders*, *Announcements*, and *Customer Login*) and also six sets of requirements in UniISExp (*General Requirements*, *Student Services*, *Services for Research and Administrative Staff*, *Library Service*, *Stream Service*, and *Research Network*) – concerning different parts of the software system – given in descriptive form. Additionally, some hints were provided with information about the concrete decisions that were expected (e.g.,

"Decide how to deal with the different channels and feed the retailer with unified messages" from Table 4). For each requirement, one or more ADDs were expected to be documented by the students.

Eventually, all participants were given access to the CoCoADvISE tool. The functionality of CoCoADvISE is described in Section 3 and the setting provided to the students can be viewed at `http://andromeda.swa.univie.ac.at/orexp` and `http://andromeda.swa.univie.ac.at/uniisexp` for ORExp and UniISExp respectively[7]. The participants of the experiment groups could reuse five ADD models which were different for the two controlled experiments. The names and descriptions of the provided ADD models are documented in Table 6 (for more details refer to the CoCoADvISE settings accessible at the aforementioned URLs.). In contrast, CoCoADvISE was modified for the participants of the control groups, so that the ADD model based support was hidden.

Blinding. In order to reduce the subjective bias of the participants and the reviewers we have applied double-blinding in both experiments. The participants were not aware of the two different treatment groups and different CoCoADvISE settings, therefore, they were not able to guess the goal of the experiments and whether they belong to an experiment or control group. In addition, the participants' documented ADDs were scrambled and given anonymized to the reviewers for evaluation. It was, thus, not possible to find out whether an ADD comes from the control or experiment group, which ADDs belong to which participants, and which ADDs belong together (i.e., were documented by the same person). Also, the reviewers did not get any other information about the goals and the different settings of the experiments.

4.4. Execution

As described in the previous section, the controlled experiments were executed in the context of the Information System Technologies and Software Architecture lectures (Faculty of Computer Science, University of Vienna) with students in the end of their bachelor studies, in Winter Semester 2013/2014 and Summer Semester 2014 respectively.

The practical course groups were randomly assigned to experiment and control groups. That is the reason why we have unequal groups of participants, namely 27 (control group) and 22 (experiment group) participants in ORExp and 56 (control group) and 66 (experiment group) participants in UniISExp. Nine students that participated in both experiments were excluded from the second experiment – although the topic and the tasks required in ORExp and UniISExp were different – in order to reduce the result bias. That led to a reduced sample of 50 and 63 participants for the control and experiment group of UniISExp respectively.

[7] To view and use the tools use the following user names (no password required): experiment or control for the experiment and control settings respectively.

ORExp	
External Interface Design	Use this reusable architectural decision model in order to decide how the service-based system will expose its interfaces to the external world.
Lifecycle Management	Use this reusable architectural decision model to decide how the lifecycle management of the remote objects (e.g., by creating per-request instances) and the resource consumption of the service-based system will be designed.
Message Routing	Use this reusable architectural decision model in order to make decisions on strategies for routing the messages in the service-based system.
Message Transformations	Use this reusable architectural decision model in order to make decisions on methods for transforming the messages inside the service-based system.
Service-based Invocations	Use this reusable architectural decision model to decide on the type of remote asynchronous invocations that will be used as well as the type of communication channels required for the exchange of messages.
UniISExp	
Application Structure	Use this reusable architectural decision model in order to decide which architectural style(s) you will use to structure your application architecture.
Data Store	Use this reusable architectural decision model in order to decide if and how you will need to store your data.
Distribute Components	Use this reusable architectural decision model in order to decide how you will distribute the different components of your system and how to design the communication between components.
Handling Processing Tasks	Use this reusable architectural decision model in order to decide how you will handle complex processing tasks required in your application.
UI Design	Use this reusable architectural decision model in order to decide how you will design the UI of your application.

Table 6: Reusable ADD Models Overview

In addition, before the execution of the controlled experiments we decided to exclude students that did not manage to achieve more than 50% of maximum points at the practical course. However, the final grade of the students at the practical course was not known at the time of execution, therefore, we excluded participants in the end of the semester and before we started with the analysis of the results. No students were excluded in ORExp while we had to exclude one student from the control group and two students from the experiment group in UniISExp. Thus, our sample for UniISExp was further reduced to 49 and 61 participants for the control and experiment group respectively.

Information about the programming experience of the students has been gathered with questionnaires that were given to the students in the beginning of the experiments. As we can see in Figure 3 and Figure 4 the programming experience, as well as the industry programming experience of the participants are quite similar for both groups. In ORExp, the control group has slightly more programming experience while in UniISExp we observe the opposite. The majority of the students in both experiments has 1 to 3 years of programming experience while some of the participants are quite experienced in programming (i.e., have more than 3 years of programming experience). However, only very few participants of both experiments have experience in industrial projects[8].

The exact same materials were handed out to all participants in the beginning of the exercise. As mentioned before, the experiment and control groups used different versions of CoCoADvISE, that is, the experiment group could use the reusable ADD models as shown in Figure 2 while this feature was deactivated for the control group. The participants had 90 minutes[9] time for reading, understanding, and performing the exercise. They were allowed to finish with the tasks earlier though. Access to the Web-based tool was given only during this session to avoid any offline work or discussion among the students. Apart from that, in order to avoid communication between the students or use of other materials the participants were prevented (i.e., using Web browser policies) from opening Web pages other than that of CoCoADvISE.

The collection of the participants' data was performed automatically during the experiment. In particular, all relevant information, such as created questionnaires, documented decisions, as well as all relevant events, such as deletions or modifications of architectural decisions, were saved in a database. In Appendix A we present six exemplary ADDs documented by the participants in the two controlled experiments along with their evaluations. The evaluation of the documented ADDs with the use of a Likert scale from 1 (lowest) to 10 (highest) was performed afterwards by two independent experts. The experts were instructed to consider the distances between the points of the 1–10 Likert scale to be constant. Hence, we argue that the corresponding variables *quality*

[8]Most of the students with industrial experience have been working from a couple of months to maximum one year at a company.

[9]This is a typical duration for a lab exercise in the two computer science courses. The experiments' tasks were thus designed given the limited time that the students would spend in the controlled environment.

and *qualityPerStudent* qualify as interval scale type. In case of high distances between the evaluations (i.e., 4 points or more between the ratings) the two experts discussed their evaluations until they reached a consensus. In total, 47 and 122 ADDs in ORExp and UniISExp respectively had to be revised. For the evaluation of the ADDs the following criteria were applied by both experts:

- The decision is stated clearly.
- The rationale of the decision is stated clearly.
- The decision corresponds to the described issue.
- The documented decision is a viable solution with regard to the described issue.
- Potential alternatives are documented.

Regarding the execution of the two controlled experiments no deviations from the initial study design occurred and no situations in which participants behaved unexpectedly.

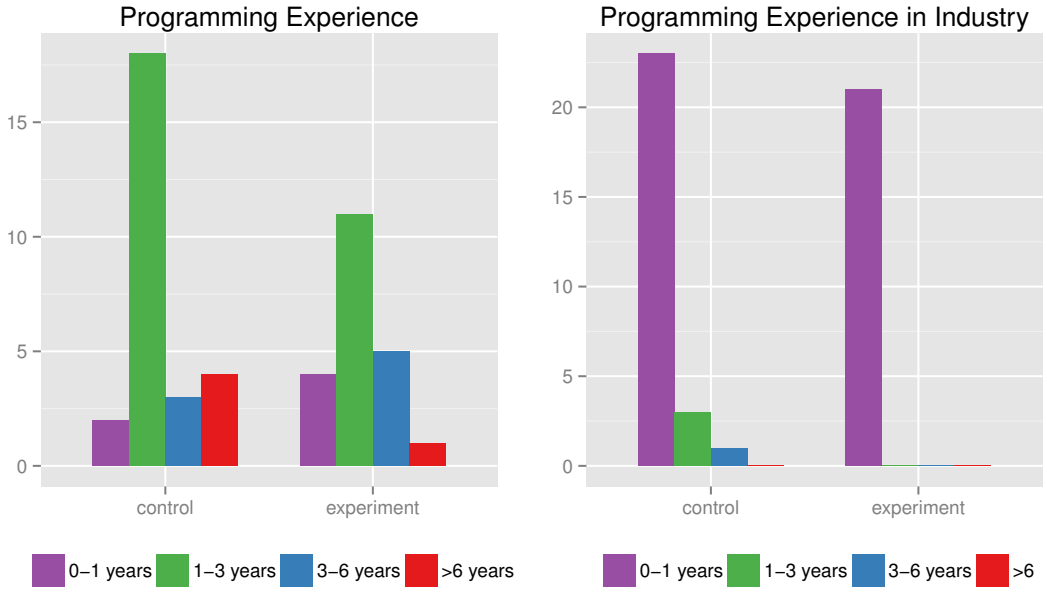


Figure 3: ORExp: Participants' Programming Experience

5. Analysis of Results

In order to analyze the data collected during the two controlled experiments and test our hypotheses (see 4.1) we used the R language and environment for statistical computing [43]. The raw data for these results are available at `http://andromeda.swa.univie.ac.at/advise-experiment/`.

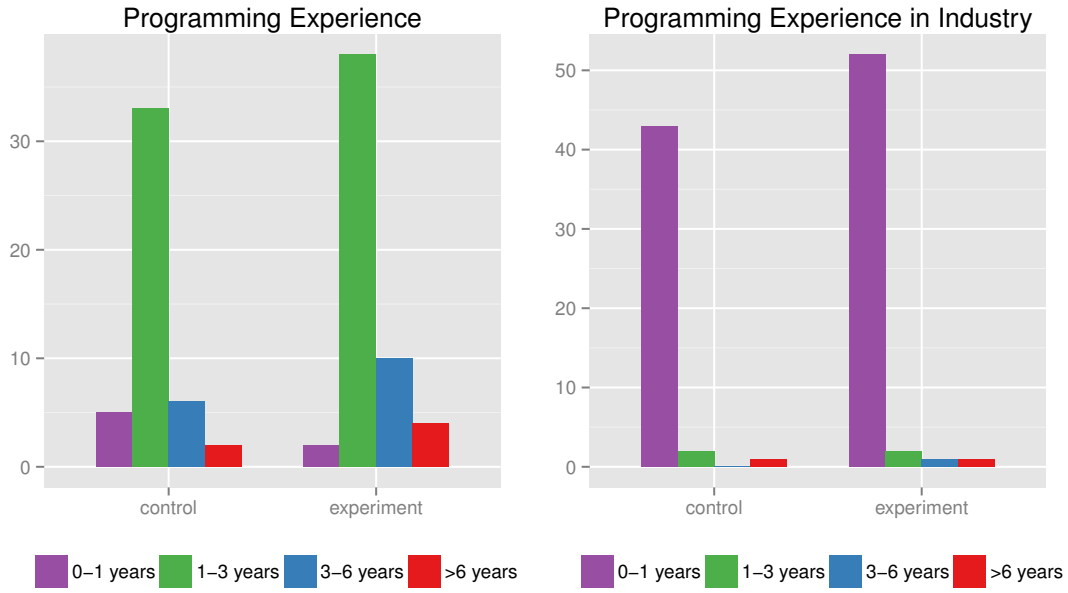


Figure 4: UniISExp: Participants' Programming Experience

5.1. Descriptive Statistics

As a first step in the analysis of the results we used descriptive statistics to compare the means and medians of the observed variables. In particular, Table 7 and Table 8 along with Figure 5 and Figure 6 display the mean and median values of the number of documented ADDs ($numOfDecisions$), the quality per ADD ($quality$), as well as the average quality of ADDs per student ($qualityPerStudent$), the time needed for completing the required tasks ($time$) and for documenting one decision ($timePerDecision$), for both control and experiment groups in the OORExp and UniISExp experiments respectively.

It is noticeable that the participants of the experiment group documented one ADD more on average than the control group: 7 against 6 and 6 against 5 for OORExp and UniISExp respectively. However, this is not necessarily an indicator of "higher efficiency" of the students as the experiment group may have needed more time for performing the tasks and thus has delivered more ADDs. We notice, though, that the control groups spent also more time on documenting a single decision, that is, 3.91 and 2.64 more minutes than the corresponding experiment groups on average. Therefore, the experiment groups needed *less time* to document *more decisions* although they have dedicated some of their time to understand and use the reusable ADD models. The reason is that CoCoADvISE with ADD model support generated semi-complete documentations which saved some time for the experiment group. We also calculated the average number of characters used for each ADD to find out whether the smaller number of ADDs and the increased time spent on their documentation was due to the smaller "size" of decisions. However, both treatment groups documented decisions of almost the same size in OORExp (489 characters for the control group

and 431 for the experiment group). We notice also a small difference in the length of the ADDs in UniISExp – 890 characters for the control group and 588 for the experiment group. This can be explained by the fact that often the participants of the control group reused parts of the pattern documentations to fill in some ADD template fields (e.g., Argumentation/Implications) which led to longer ADD texts – unlike for the experiment group these documentations provided the main source for decision support.

Variable	**Means**		**Medians**	
	control	**exp.**	**control**	**exp.**
numOfDecisions	5.59	7.57	6	7
quality	4.82	5.46	4.85	5.29
qualityPerStudent	4.9	5.35	4.5	5.5
time (min)	44.71	35.75	48.11	36.69
timePerDecision (min)	9.06	5.15	8.12	4.79

Table 7: Means and Medians of Observed Variables for ORExp

As mentioned in Section 4, each documented ADD was evaluated by two independent experts using a 10-point Likert scale ranging from very low quality (1) to very high quality (10). Likert scales are treated in the literature as both ordinal (e.g., [44]) and interval (e.g., [45]) scales. Ordinal scales show us the order, but not the distances between the ranking, that means, that in a 10-point Likert scale 2 means better quality than 1, 3 better quality than 2, and so on. Interval scales, on the other hand, show the order of things with equal intervals between the scale points. In order to be able to use descriptive statistics and statistical tests to analyze the quality of decisions we must treat the 10-point Likert scale as an interval scale. We assume, therefore, that this holds true in our case.

The average quality of the experiment group's ADDs is 5.46 and 6.96 compared to 4.82 and 6.34 in the control group for the ORExp and UniISExp respectively. In addition, the students in the experiment group delivered ADDs of better quality on average. To summarize, we would say that the treatment group that used the ADD models support *documented more ADDs* and achieved results of *better quality* and in *less time*.

On average, the participants of the experiment groups used 7 (ORExp) and 6 reusable ADD models. In addition, we calculated the time needed for filling in the questionnaires and the ADD templates separately. In ORExp 25% and in UniISExp 16% of the total time (1.07 and 0.81 minutes per questionnaire respectively) was spent on answering the questionnaires. The rest of the time was spent on filling in the semi-complete ADD documentations. Therefore, the effort for using the decision support capabilities of CoCoADvISE is very small in comparison with the effort needed to make and document decisions without automated or semi-automated guidance.

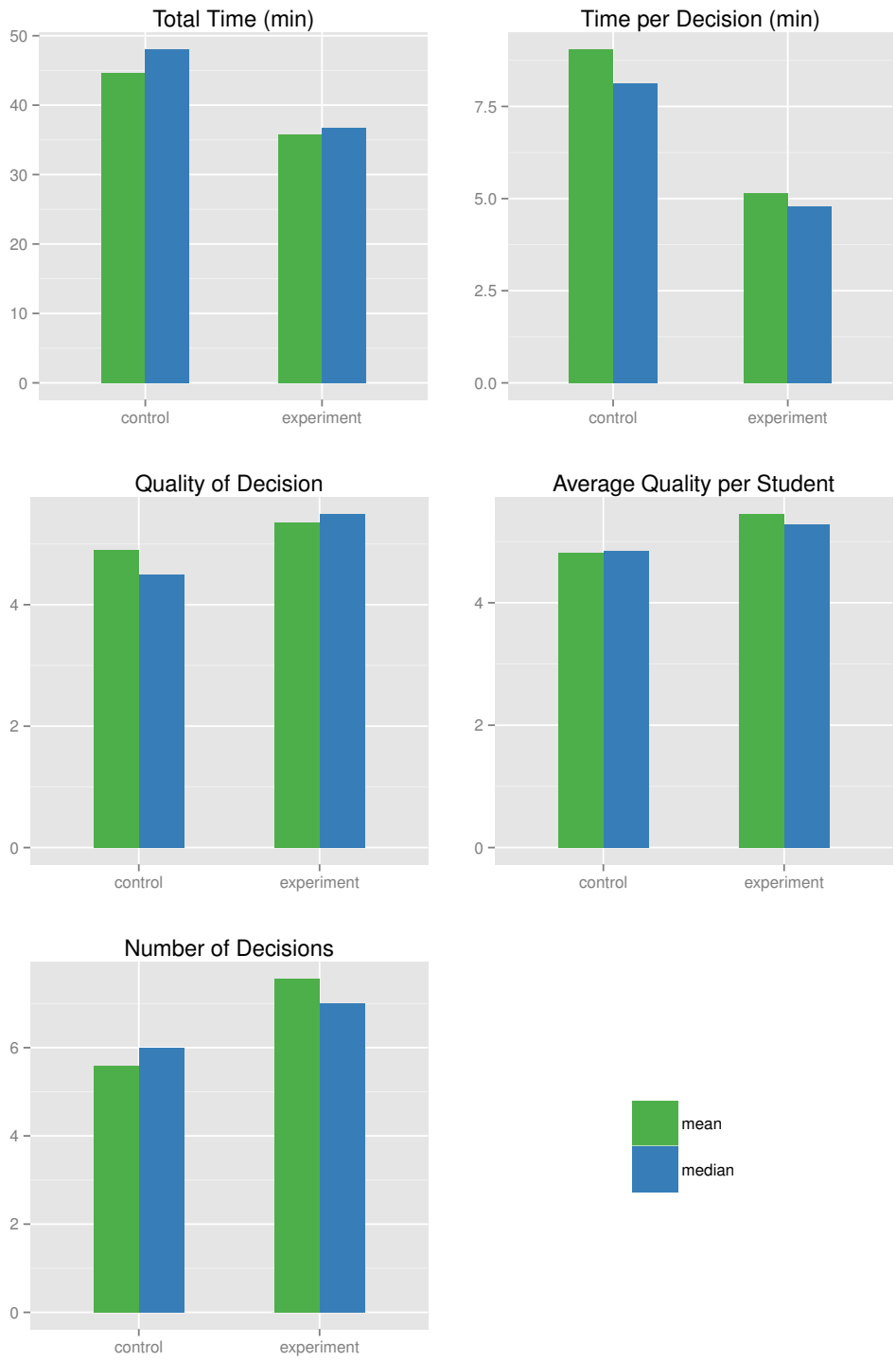


Figure 5: ORExp: Comparison of Means and Medians for the Observed Variables

5.2. Data Set Reduction

The few outliers that we noticed by studying the deviations from the means for each of the observed variables correspond to different participants, that is,

Variable	Means		Medians	
	control	**exp.**	**control**	**exp.**
numOfDecisions	5.70	6.72	5	6
quality	6.34	6.96	6.79	7.04
qualityPerStudent	6.37	6.92	7	7
time (min)	57.50	44.99	58.37	45.74
timePerDecision (min)	10.72	8.08	11.52	7.20

Table 8: Means and Medians of Observed Variables for UniISExp

these outliers correspond to ADDs documented by different students. Thus, the fact that these points have much higher or much lower values than the means (e.g., a student documented only a few decisions or needed much time for one decision) does not make necessarily the participant an outlier. Therefore, we excluded only students from the second controlled experiment (UniISExp) that had participated in the first as well and students who did not complete the practical course successfully (i.e., scoring less than 50% of the maximum points), and who therefore would make the study results vulnerable. This was done, however, before the data analysis (see explanation in Section 4.4); at this stage, we did not perform any further data set reduction.

5.3. Hypotheses Testing

5.3.1. Testing for Normal Distribution

Parametric tests like the *t*-test assume the normal distribution of the analyzed data. In order to decide whether to use parametric or non-parametric test for the analysis of the data, we applied the Shapiro-Wilk [46] normality test. The null hypothesis of the Shapiro-Wilk test states that the input data is normally distributed. We test the normality at a level of confidence of 95%. That means that if the calculated p-value is lower than 0.05 the null hypothesis is rejected and the input data is *not* normally distributed. Conversely, if the p-value is higher than 0.05, we can not reject the null hypothesis that the data is normally distributed.

Table 9 lists the p-values of the Shapiro-Wilk normality test for each observed variable and treatment group (all values are rounded to 4 decimal digits) for both controlled experiments. P-values that indicate normal distribution are emphasized (i.e., using **bold** font). We can see that only *numOfDecisions* and *qualityPerStudent* for ORExp and *time* for UniISExp controlled experiment exhibits a tendency of being normally distributed, while for the other variables we can not assume that they are normally distributed. As a result, we decided to pursue a non-parametric statistical test for analyzing the data.

5.3.2. Comparing the Means of Variables

To compare the means of the observed variables for the control and experiment groups, we applied the one-tailed Wilcoxon rank-sum test [47], a non-

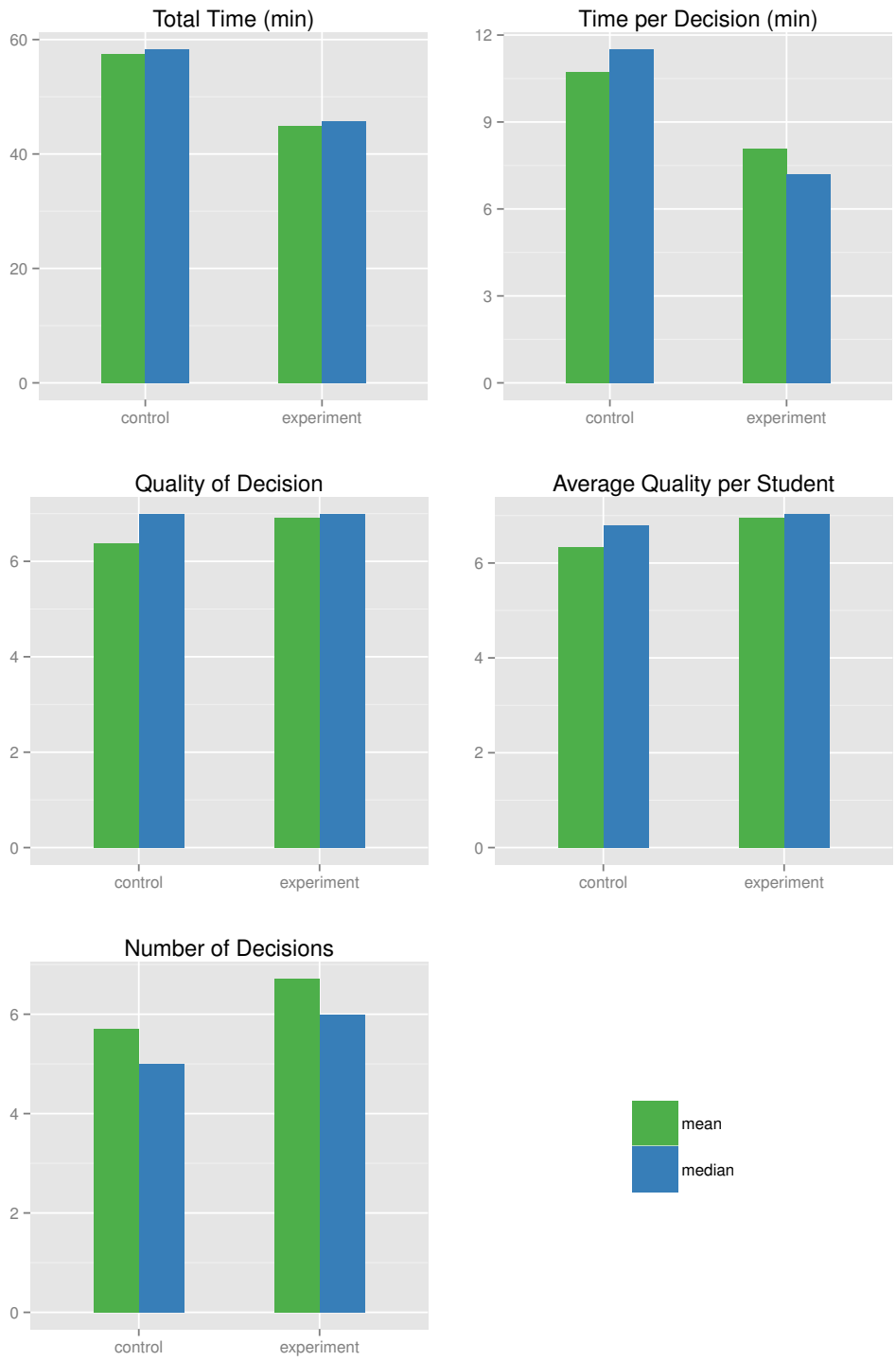


Figure 6: UniISExp: Comparison of Means and Medians for the Observed Variables

parametric test for assessing whether one of two data samples of independent observations is stochastically greater than the other. Its null hypothesis, which is appropriate for the hypotheses in our experiments, is that the means of the first

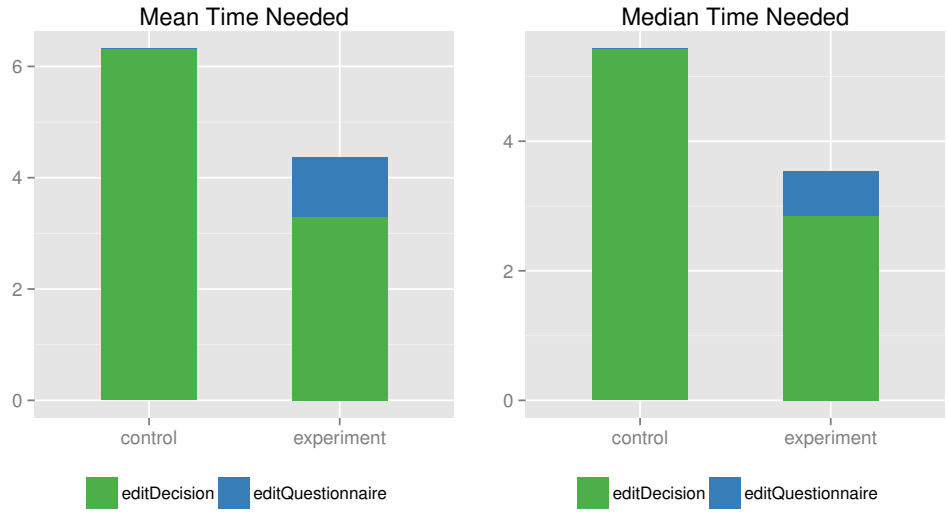


Figure 7: ORExp: Comparison of Time Spent by Participants for Making and Documenting ADDs

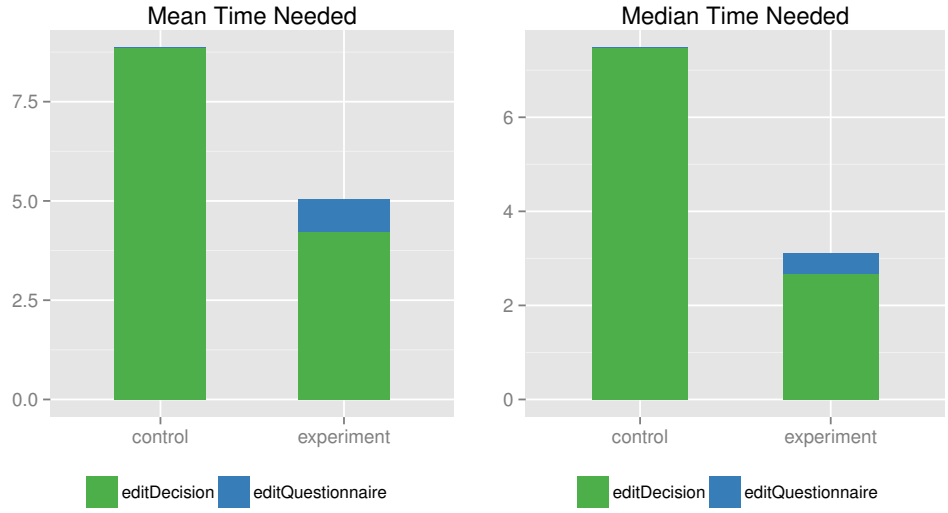


Figure 8: UniISExp: Comparison of Time Spent by Participants for Making and Documenting ADDs

variable's distribution is less than or equal to the means of the second variable's distribution, so that we can write $H_0 : A \leq B$. The Wilcoxon rank-sum test tries to find a location shift in the distributions, i.e., the difference in means of two distributions. The corresponding alternative hypothesis H_A could be written as $H_A : A > B$. If a p-value for the test is smaller than 0.05 (i.e., the level of confidence is 95%), the null hypothesis is rejected and the distributions are shifted. If a p-value is larger than 0.05, the null hypothesis can not be rejected, and we can not claim that there is a shift between the two distributions.

Table 10 contains the p-values of five Wilcoxon rank-sum tests that were performed to find out whether we can reject the null hypotheses presented in

Variable	*OrExp* **p-Value**		*UniISExp* **p-Value**	
	control	**exp.**	**control**	**exp.**
numOfDecisions	**0.493**	**0.0941**	0.0009	0.0042
quality	$0(10^{-5})$	$0(10^{-5})$	$0(10^{-11})$	$0(10^{-9})$
qualityPerStudent	**0.7323**	**0.7145**	0.0046	**0.0643**
time (min)	0.0107	**0.8832**	**0.3593**	**0.2717**
timePerDecision (min)	0.002	**0.918**	**0.5181**	$0(10^{-10})$

Table 9: Shapiro-Wilk Normality Test

Section 4.1. Note that only the first four decimal places of the results are reported. Based on the obtained p-values, we can assess that almost all distributions show a statistically significant shift between each other and that most of the null hypotheses can be rejected. Analogously, Table 11 contains the p-values of t-Tests for those variables, where the assumption of normal distribution could not be rejected (see Table 9).

Testing Hypothesis $\mathbf{H_1}$. The experiment group documented significantly more ADDs than the control group. We reject the null hypothesis H_{01} (i.e., the use of reusable ADD models has no effect on the quantity related effectiveness of software architecture students) since the calculated p-value is 0.0027 (t-test: 0.0021) and 0.008 for the OrExp and UniISExp experiments respectively. Hence, there is evidence that the use of reusable ADD models for decision making and documentation increases the *quantity related effectiveness* of software architecture students.

Testing Hypothesis $\mathbf{H_2}$. In our experiments, we observed that the participants of the experiment groups delivered ADDs of better quality than the participants of the control groups. As this observation holds for both variables *quality* and *qualityPerStudent* we tested the null hypothesis H_{02} (i.e., the use of reusable ADD models has no effect on the quality related effectiveness of software architecture students) for these two variables. In the experiment OrExp, the p-values 0.0332 and 0.055 (> 0.05) do not allow us to reject H_{02} completely (however, t-test: 0.0330). We can reject this hypothesis for the quality of single ADDs but not for the average quality of documented ADDs per student. For UniISExp though, in which we tested a bigger sample of ADDs and participants, H_{02} could be rejected given the p-values 0.0459 and 0.0173 for the variables *quality* and *qualityPerStudent* respectively. Therefore, there is evidence for supporting H_2, that is, the ADDs of the experiment group were in total of better quality – according to the reviewers' evaluations – than those of the control group and the single participants' performance was also significantly "better". Hence, we can report evidence that using reusable ADD models for making and documenting decisions also increases the *quality related effectiveness* of its users.

Testing Hypothesis $\mathbf{H_3}$. Finally, we discovered that the experiment group needed significantly less time to document a decision. This holds also for the total time this group spent on the assigned tasks. With a p-value of 0.0045 and 0.0001 for *time* and *timePerDecision* in OrExp and corresponding values close to 0 (10^{-5}) in UniISExp we can reject the null hypothesis H_{03} for both experiments (the same holds for the corresponding t-test for the variable *time*), that is, the use of reusable ADD models has no effect on the time related efficiency of software architecture students. Thus, we can conclude that there is evidence that reusable ADD models lead to increased *time related efficiency* of software architecture students as well.

Hypothesis	**Variable** (μ)	**p-Value**	
(**Assumption**)		**ORExp**	**UniISExp**
$\mathbf{H_{01}}$ ($\mu_{exp} \geq \mu_{control}$)	$numOfDecisions$	0.0027	0.0080
$\mathbf{H_{02}}$ ($\mu_{exp} \geq \mu_{control}$)	$quality$	0.0332	0.0459
	$qualityPerDecision$	0.0550	0.0173
$\mathbf{H_{03}}$ ($\mu_{exp} \leq \mu_{control}$)	$time$	0.0045	$0(10^{-5})$
	$timePerDecision$	0.0001	$0(10^{-5})$

Table 10: Hypotheses Testing Results (Wilcoxon rank-sum Test)

Hypothesis	**Variable** (μ)	**p-Value**	
(**Assumption**)		**ORExp**	**UniISExp**
$\mathbf{H_{01}}$ ($\mu_{exp} \geq \mu_{control}$)	$numOfDecisions$	0.0021	
$\mathbf{H_{02}}$ ($\mu_{exp} \geq \mu_{control}$)	$qualityPerDecision$	0.0330	
$\mathbf{H_{03}}$ ($\mu_{exp} \leq \mu_{control}$)	$time$		$0(10^{-6})$

Table 11: Hypotheses Testing Results (t-Test)

6. Discussion

The following subsections discuss our main findings and their implications and inferences for software architects. We also report on the threats to validity.

6.1. Evaluation of Results and Implications

Increased Effectiveness. The first two hypotheses, i.e., H_1 and H_2, are related to the effectiveness of software architecture students which we study separately with regard to the quantity and the quality of the documented ADDs. As reported in Section 5, we have provided strong evidence for rejecting the corresponding null hypotheses H_{01} and H_{02}. Thus, reusable ADD models contribute both to the completeness and the quality of ADD documentations.

The semi-automated guidance using questionnaires provided by the CoCoADvISE tool allowed software architecture students (representative for novice software architects) to (1) identify the ADDs that had to be made, (2) understand the design space in the corresponding case study, (3) find out the alternatives for each design issue, and finally (4) document the appropriate design solution according to the requirements. Some of the participants of the experiment groups stated afterwards that CoCoADvISE's reusable models helped them find quickly the answer to their problem without needing to read all design pattern documentations. Especially, that turned out to be useful in cases where the design solutions were not so obvious from the requirements and required a better research and analysis of the design situation. For instance, for handling some complex processing tasks in the UniISExp case study, "single or multi threaded implementation of the tasks" would be viable solutions inferred by the reusable ADD models. Yet, most of the participants of the control group opted for a "pipes and filters" solution, which would clearly be "overkill" in this specific situation. Another phenomenon that we observed was that participants of the control groups were often not confident about what they should document in the ADD template fields. On the contrary, the reusable ADD models provided support (i.e., some fields are filled in automatically) and guidance (i.e., the reusable ADD models already contain design rationale) for filling in the decision related fields. We decided not to fill in further fields automatically (e.g., positions, arguments, etc.) as this would give a big advantage to the experiment groups and would make the comparison between the treatment groups difficult.

The observed phenomenon of experiment groups documenting more decisions can possibly be explained as follows. The reusable ADD models guided the students to follow-on decisions that were not considered by the participants of the control groups at all.

Systematically capturing and leveraging reusable AK in the form of reusable ADD models therefore may lead to increased effectiveness of architects. Our findings also validate Zimmermann et al.'s claim about some of the benefits of architecting with reusable ADD models: reusable ADD models (1) provide means for the semi-automatic decision identification and (2) improve the quality of decision making [8].

Increased Efficiency. We showed in the previous section that our last alternative hypothesis H_3 could also be accepted, that is, using CoCoADvISE's support on reusable ADD models reduces time and effort for software architecture students. This finding was highly expected though, as much "manual" work for making and documenting ADDs (e.g., reading documentations, filling in ADD templates repeatedly) is made redundant due to the semi-automated guidance by the questionnaires and the semi-complete documentations generated from decision recommendations. Thus, architects may need less time to document ADDs when guided by reusable ADD models.

6.2. Threats to Validity

To ensure the validity of our results, we consider the categorization of validity threats of Wohlin [42] and discuss each of them separately in the context of our controlled experiments.

Conclusion Validity. The conclusion validity focuses on the relationship between the treatment we used in the experiment and the actual outcome, i.e., on the existence of a significant statistical relationship. The way we measured the working time of the students automatically from the database entries may pose a threat to conclusion validity, as the users might have spent some working time idle or with pen and paper. Apart from that, to measure the actual time spent on working with CoCoADvISE for performing the assigned tasks is very difficult, if not impossible, as the participants may have spent some time reading the tasks or familiarizing with the tool. However, we think that idle working times, times spent on other tasks, or offline work can largely be excluded due to the limited experiment time of 90 minutes in which the participants needed to work in a concentrated manner in order to get the work done.

In addition, the interpretation of the 10-point Likert scale that was used to rate the ADDs may pose a threat to the conclusion validity. In Section 5, we argued that we consider the Likert scale an interval scale, and thus the descriptive statistics and statistical tests that we applied are, making this assumption, valid. Another potential threat to validity is the subjectivity of the quality ratings and of the reviewers. It could have happened as well that the one reviewer evaluated more strictly than the other. To reduce these risks, we asked two independent experts in the field of software architecture to evaluate all ADDs for both experiments and calculated afterwards the quality of each ADD on average. In case of disagreements in the evaluations the two reviewers needed to discuss their ratings until they reached a consensus. Nevertheless, this does not erase the subjectivity of the evaluations completely as some aspects related to the quality of ADDs like the evaluation of ADDs after the implementation of the software system are not taken into consideration in our case.

Internal Validity. The internal validity refers to the extent to which treatment or independent variables caused the effects seen on the dependent variables. In order to reduce this kind of validity threats, we made sure that the participants of both groups had at least medium experience in programming and design – with slight differences – and that they were aware of the architectural design patterns they had to use for making and documenting architectural decisions. For this, the participants were asked to study all related patterns in advance. Apart from this, a few patterns were applied in previous practical exercises (i.e., programming exercises concerning the implementation of various software systems) of the lecture by the students.

The experiments were carried out in controlled environments and the treatment group members were in different rooms. Thus, they could not know the goals of the experiment, as they were not aware of the existence of different groups or/and the different CoCoADvISE settings for the two treatment groups.

During the experiment, the students were allowed to use only the CoCoADvISE tool and all other applications and Web pages were blocked. This way, we prevented access to other Internet materials as well as the participants' communication through chat applications. Also, an observer in the room ensured that no interactions between the participants of the same room occurred to ensure that the students worked individually.

We also prevented any access to CoCoADvISE and the accompanying materials outside the dedicated session or outside the labs where the controlled experiments were carried out.

Construct Validity. The construct validity focuses on the suitability of the experiment design for the theory behind the experiment and the observations.

The variables that have been observed in the experiment are regarded accurate and objective as they are related to the actual use of tools and were automatically extracted from the database. The students did not have any training or experience with the tools. A potential threat to validity may be posed by improper use of CoCoADvISE by participants who did either not understand the purpose of the reusable ADD models and the questionnaires or did not comprehend the tasks. We also need to take into account language difficulties in some cases (CoCoADvISE texts were in English, all other materials were both in English and German). We tried to overcome such problems by encouraging the students to ask the instructors for further explanations. Another potential threat to construct validity is the fact that only one measure was used to evaluate the quality of the documented ADDs, which does not allow us to cross-check the experiments' results.

Finally, there is a potential threat to validity imposed by the participating subjects, knowing the research topics of our research group and therefore being able to "guess" results that we hoped to obtain. We tried to minimize the impact of this threat by choosing independent and external experts that were not aware of our research agenda.

External Validity. The external validity is concerned with whether the results are generalizable outside the scope of our study. The subjects of the experiments had medium programming experience and were familiar with the ADDs they were asked to make. However, only few students had experience in the industry. We hence consider the participants of both controlled experiments to be representative for novice software developers or architects. A threat to validity of the relevance of our study's findings in practice is the use of students instead of practitioners in both experiments. We tried to mitigate the threat by educating the students on the patterns and architectural decisions used in the experiment. At the time of the experiments, all participants had theoretical knowledge of, as well as programming experience with the majority of the design solutions that had to be considered for the required tasks. Therefore, our results can be likely to a certain extent transferred to more experienced architects. However, we will need to conduct similar experiments with practitioners in order to be able to consider our experiment results generalizable – applicable to professional

novice (or more experienced) software architects. In addition, Kitchenham et al. regard students close to practitioners, as they are considered to be the next generation of software professionals [41], which strengthens our claim that our findings could apply to professional architects as well.

Also, the fact that the treatment groups used the CoCoADvISE tool only, poses the threat that the results are specific only for this tool. However, we tried to implement in CoCoADvISE general concepts and practices from reusable ADD models [3] and ADD documentations [16]. Hence, we expect the same results if the participants worked with tools based on similar concepts as well.

As mentioned before, the measurements were extracted from the tool database, avoiding any bias by the experimenters.

6.3. Inferences

van Heesch et al. found out that the quality of ADDs increases when focus is set on the use of design patterns for documentation [12]. Our findings confirm and supplement these previous results for students of software architecture. That means that the use of design patterns in the form of reusable ADD models further increases the effectiveness of students leading to less effort and better quality for ADD documentations. We claim that our results can be transferable to novice/trainee software architects, that needs to be proven, however, in an industrial context with practitioners.

Although we have conducted our experiments with students with software architecture background and with little commercial experience, we believe that our results can apply similarly to novice or more experienced architects in the industry. To validate these claims we will need more empirical evidence and feedback from the use of reusable architectural decision models in practice and in the context of industrial projects.

In practice, ADDs remain either undocumented or partially documented and the documentations are often dispersed in different places like wikis, meeting minutes, and issue tracking systems [40]. In addition, documenting ADDs that correspond to recurring design issues is tedious and time-consuming. Providing the means for semi-automated decision making and documentation based on reusable ADD models may encourage software architects to capture AK regularly and systematically. That will contribute to reducing AK vaporization [2].

It is of course important that reusable ADD models that have been documented in the literature (e.g., [8, 9, 21, 22]) and have been tested in a very limited scale (i.e., by practitioners of the same company, etc.) get validated and enriched by practitioners from different domains. That will increase the acceptance of such reusable AK and confirm Nowak et al.'s vision of collaboration and knowledge exchange between different AK repositories (i.e., repositories containing reusable architectural decision models) [48]. Therefore, the usefulness of tools like CoCoADvISE which provide semi-automated decision support based on such ADD models will get more significant.

7. Conclusions and Future Work

In this paper, we reported the results of two separate controlled experiments on architectural decision making and documentation based on reusable ADD models. Due to the current lack of empirical evidence, the experiments were designed to determine the supportive effect of reusable ADD models for software architects. We were particularly interested in quantifying their impact on a software architecture student's efficiency and effectiveness while making and documenting decisions. Hence, we implemented and instrumented a custom-made tool (CoCoADvISE) for making and documenting ADDs, that tries to incorporate the best practices of current (i.e., state of the art) ADD tools, in such a way that each action in the tool was timestamped and stored in a database. This valuable information allowed us to extract and deduce efficiency and effectiveness related metrics.

Our experiments provided strong evidence that reusable ADD models can potentially increase both the effectiveness and the efficiency of their users while making and documenting ADDs. We can further report, that our findings are in line with similar studies (see, e.g., [12]) and support the claims regarding reusable ADDs (see, e.g., [8, 9, 21, 22]) in principle. We consider our results to be applicable for novice software architects as well. Thus, as part of our ongoing future work, we will repeat our experiments with different reusable ADD models and different types of participants. In particular, we strive for experimenting with practitioners to see if our assumptions and results are still valid in industrial contexts. At the same time, we plan to further investigate the educational aspect of reusable ADD models. To that end we want to find out if and to what extent reusable ADD models are conducive for novice software architects to learning how to systematically use patterns.

Acknowledgments

We would like to thank the reviewers who evaluated the architectural decisions in both controlled experiments. We also thank all students for participating in the experiments.

Appendix A. Examples of Documented ADDs

We present in this appendix exemplary ADDs documented by the participants of the two controlled experiments along with the experts' ratings for these decisions. In particular, we include three decisions from ORExp and three decisions from UniISExp.

Name	D01: Use Fire and Forget for Logging
Group	Communication/Logging
Issue	All messages that the online retailer exchanges are sent to a logging system that stores them in a database for 1 year.
Decision	Fire and Forget Pattern
Assumptions	Loss of logging messages is acceptable.
Positions	Our logging service is implemented as a remote object. Recording of log messages must not influence the execution of the client and loss of individual log messages is acceptable. In this case Fire and Forget can be used.
Arguments/ Implications	A client application wants to notify a remote object for an event. Neither a result is expected, nor does the delivery have to be guaranteed. A one-way exchange of a single message is sufficient. -> simple, non-blocking communication
Related Decisions	-

Table A.1: Example 1 from OORExp / Reviewer 1: 7, Reviewer 2: 8

Name	Process Orders - Multiple single orders
Group	Messaging
Issue	Inventory systems can only process single orders.
Decision	Splitter
Assumptions	The Splitter splits the order into individual order items. Each message refers to the original message (e.g., the order ID) so that the processing results can be correlated back to the original message.
Positions	-
Arguments/ Implications	A Splitter can break large messages into individual messages to simplify further processing. In our Systems this needs to be done because the inventory systems can only process single orders. The only downside is that the overhead gets bigger because of the correlation identifiers.
Related Decisions	Process Orders - Routing and Take Orders.

Table A.2: Example 2 from OORExp / Reviewer 1: 7, Reviewer 2: 5

Name	OrderMessageLocationProcessor-ADD1: Message processing
Group	Message Routing and Processing
Issue	OrderMessageLocationProcessor ADD1: Message processing
Decision	Use a message splitter
Assumptions	The order processing sever need to receive orders from different locations, the call center and the website. The order processing server needs to know where is one order coming from and treat all orders equally.
Positions	Message splitter with Message aggregator
Arguments/ Implications	Each order message contents the location information and the order information, before processing the order, a splitter is needed to split the order message. As the retailer wants to treat each order equally, an aggregator is not needed here.
Related Decisions	OrderConfirmMessageRouter-ADD2: Message routing/filtering

Table A.3: Example 3 from OORExp / Reviewer 1: 10, Reviewer 2: 9

Name	Decision 02 S-T
Group	Streaming
Issue	Streams should be kept on a central server and be downloaded on demand, bandwidth and transformation in different audio/video formats is a problem.
Decision	I have chosen asynchronous streaming where the sequence of the packets is important, but not the time.
Assumptions	Because of the high amount of students it is important to keep the bandwidth low when streaming. Because of that, the streaming should be asynchronous, because it is faster and needs less bandwidth as synchronous or isochronous streaming.
Positions	Alternatives would be synchronous or isochronous streaming, which are "better" but also need a lot more resources, especially when streaming to multiple clients like it is done at the university.
Arguments/ Implications	Streaming offers high scalability as it requires less memory usage at the server side. Also, it is simple to implement and maintain. However, it cannot support transactional processing or interaction between different clients. And as I chose the asynchronous streaming I can decrease the memory usage even more.
Related Decisions	none

Table A.4: Example 4 from UniISExp / Reviewer 1: 9, Reviewer 2: 9

Name	Communication between the University-IS and e-learning
Group	Communication between the uni-IS and the e-learning
Issue	Current system: bad communication between the uni-is and the e-learning and the information not in time.
Decision	Shared repository
Assumptions	Invocation parameters are inefficient for large data sets also bad if the information varies from invocation to invocation.
Positions	Very often data needs to be shared between components. In sequential architectures the only way to share data between the components is to use invocation parameters but this is inefficient for large data sets. Also it might be inefficient, if the shared information varies from invocation to invocation. Finally the long-term persistence of the data requires a centralized data management. Thus, Shared Repository is used as a central data store, accessed by all other independent components. It offers suitable means for accessing the data, for instance, a query API or language and is scalable to meet the clients' requirements. It must ensure data consistency and must handle problems of resource contention, for example by locking accessed data. It might also introduce transaction mechanisms. A Shared Repository maintains the common data on which the application's components operate, the components themselves access and modify the data in the shared repository, and the state of the data in the Shared Repository instigates the control flow of specific components.
Arguments/ Implications	The Shared Repository pattern increases the level of abstraction in the code. This may decrease understandability for developers who are unfamiliar with the pattern. Although implementing the pattern reduces the amount of redundant code, it generally increases the number of classes that must be maintained. The Shared Repository pattern helps to isolate both the service and the list access code. Isolation makes it easier to treat them as independent services and to replace them with mock objects in unit tests. Typically, it is difficult to unit test the repositories themselves, so it is often better to write integration tests for them. If the data is being cached in heavily loaded systems, performance can be an issue.
Related Decisions	Faster, better for large data and long term persistence of the data requires, suitable for accessing the data

Table A.5: Example 5 from UniISExp / Reviewer 1: 6, Reviewer 2: 5

Name	D01: Architectural style for distribution
Group	System Design
Issue	System needs to be deployed on different machines
Decision	Design of a service-oriented architecture
Assumptions	(1) System needs to be scalable and deployed onto several different machines. (2) Heterogenous components like e-learning platforms (that change very fast), research networks and library components need to be integrated in such a way that changing or replacing one of the subsystems does not lead to a huge effort in reprogramming the core application.
Positions	A poor alternative to this approach would be to implement each functionality on its own and just access a shared repository where input/output data is stored.
Arguments/ Implications	Service-based architecture is a very well established pattern and meets all the requirements for this project. As there is a very loose coupling between the communicating entities, it is no problem to replace or add subsystems which is very crucial for such a dynamic system.
Related Decisions	-

Table A.6: Example 6 from UniISExp / Reviewer 1: 8, Reviewer 2: 9

References

[1] A. Jansen, J. Bosch, Software Architecture as a Set of Architectural Design Decisions, in: 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), Pittsburgh, PA, USA, IEEE Computer Society, 2005, pp. 109–120.

[2] N. B. Harrison, P. Avgeriou, U. Zdun, Using Patterns to Capture Architectural Decisions, IEEE Software 24 (4) (2007) 38–45.

[3] O. Zimmermann, T. Gschwind, J. Küster, F. Leymann, N. Schuster, Reusable Architectural Decision Models for Enterprise Application Development, in: 3rd International Conference on Quality of Software Architectures (QoSA), Medford, MA, USA, Springer, 2007, pp. 15–32.

[4] R. Farenhorst, R. Izaks, P. Lago, H. Van Vliet, A Just-In-Time Architectural Knowledge Sharing Portal, in: Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA), 2008, pp. 125–134.

[5] M. Galster, M. A. Babar, Empirical Study of Architectural Knowledge Management Practices, in: IEEE/IFIP Conference on Software Architecture (WICSA), 2014, pp. 239–242.

[6] O. Zimmermann, U. Zdun, T. Gschwind, F. Leymann, Combining Pattern Languages and Reusable Architectural Decision Models into a Comprehensive and Comprehensible Design Method, in: 7th Working IEEE/IFIP Conference on Software Architecture (WICSA), Vancouver, BC, Canada, IEEE Computer Society, 2008, pp. 157–166.

[7] D. Falessi, G. Cantone, R. Kazman, P. Kruchten, Decision-making Techniques for Software Architecture Design: A Comparative Survey, ACM Computing Survey 43 (4) (2011) 33:1–33:28.

[8] O. Zimmermann, J. Koehler, L. Frank, Architectural Decision Models as Micro-Methodology for Service-Oriented Analysis and Design, in: D. Lübke (Ed.), Proceedings of the Workshop on Software Engineering Methods for Service-oriented Architecture (SEMSOA 2007), Hannover, Germany, 2007, pp. 46–60.

[9] I. Lytra, S. Sobernig, U. Zdun, Architectural Decision Making for Service-Based Platform Integration: A Qualitative Multi-Method Study, in: Proceedings of the 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, WICSA-ECSA'12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 111–120.

[10] S. Abrams, B. Bloom, P. Keyser, D. Kimelman, E. Nelson, W. Neuberger, T. Roth, I. Simmonds, S. Tang, J. Vlissides, Architectural Thinking and Modeling with the Architects' Workbench, IBM Systems Journal 45 (3) (2006) 481–500.

[11] J. F. Hoorn, R. Farenhorst, P. Lago, H. van Vliet, The Lonesome Architect, Journal of Systems and Software 84 (9) (2011) 1424–1435.

[12] U. van Heesch, P. Avgeriou, U. Zdun, N. Harrison, The supportive effect of patterns in architecture decision recovery – A controlled experiment, Science of Computer Programming 77 (5) (2012) 551–576.

[13] O. Zimmermann, Architectural Decisions as Reusable Design Assets, IEEE Software 28 (1) (2011) 64–69.

[14] I. Lytra, H. Tran, U. Zdun, Supporting Consistency Between Architectural Design Decisions and Component Models Through Reusable Architectural Knowledge Transformations, in: Proceedings of the 7th European Conference on Software Architecture (ECSA), ECSA'13, Springer-Verlag, Berlin, Heidelberg, 2013, pp. 224–239.

[15] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, R. Little, Documenting Software Architectures: Views and Beyond, Pearson Education, 2002.

[16] J. Tyree, A. Akerman, Architecture Decisions: Demystifying Architecture, IEEE Software 22 (2) (2005) 19–27.

[17] L. Lee, P. Kruchten, Capturing Software Architectural Design Decisions, in: 2007 Canadian Conference on Electrical and Computer Engineering, IEEE Computer Society, 2007, pp. 686–689.

[18] M. Shahin, P. Liang, M. R. Khayyambashi, Architectural design decision: Existing models and tools, in: IEEE/IFIP Conference on Software Architecture/European Conference on Software Architecture, IEEE, 2009, pp. 293–296.

[19] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, M. A. Babar, A comparative study of architecture knowledge management tools, Journal of Systems and Software 83 (3) (2010) 352–370.

[20] D. Tofan, M. Galster, P. Avgeriou, W. Schuitema, Past and future of software architectural decisions – A systematic mapping study, Information and Software Technology 56 (8) (2014) 850–872.

[21] U. Zdun, M. Strembeck, Reusable Architectural Decisions for DSL Design: Foundational Decisions in DSL Development, in: Proceedings of 14th European Conference on Pattern Languages of Programs (EuroPLoP 2009), Irsee, Germany, 2009, pp. 1–37.

[22] C. Mayr, U. Zdun, S. Dustdar, Reusable Architectural Decision Model for Model and Metadata Repositories, in: F. de Boer, M. Bonsangue, E. Madelaine (Eds.), Formal Methods for Components and Objects, Vol. 5751 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009, pp. 1–20.

[23] A. MacLean, R. Young, V. Bellotti, T. Moran, Questions, Options, and Criteria: Elements of Design Space Analysis, Human-Computer Interaction 6 (1991) 201–250.

[24] M. A. Babar, I. Gorton, A Tool for Managing Software Architecture Knowledge, in: Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent, SHARK-ADI'07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 11–.

[25] R. Capilla, F. Nava, J. C. Duenas, Modeling and Documenting the Evolution of Architectural Design Decisions, in: Proceedings of the Second Workshop on SHAring and Reusing Architectural Knowledge Architecture, Rationale, and Design Intent, SHARK-ADI'07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 9–.

[26] A. Zalewski, S. Kijas, D. Sokołowska, Capturing Architecture Evolution with Maps of Architectural Decisions 2.0, in: Proceedings of the 5th European Conference on Software Architecture, ECSA'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 83–96.

[27] M. Shahin, P. Liang, M. R. Khayyambashi, Improving Understandability of Architecture Design Through Visualization of Architectural Design Decision, in: Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge, SHARK'10, ACM, New York, NY, USA, 2010, pp. 88–95.

[28] A. Jansen, J. V. D. Ven, P. Avgeriou, D. K. Hammer, Tool Support for Architectural Decisions, in: Proceedings of the 6th working IEEE/IFIP Conference on Software Architecture, IEEE Comp. Soc., 2007, pp. 4–4.

[29] M. Konersmann, Z. Durdik, M. Goedicke, R. H. Reussner, Towards Architecture-centric Evolution of Long-living Systems (the ADVERT Approach), in: Proceedings of the 9th International ACM SIGSOFT Conference on Quality of Software Architectures, QoSA'13, ACM, New York, NY, USA, 2013, pp. 163–168.

[30] M. Nowak, C. Pautasso, Team Situational Awareness and Architectural Decision Making with the Software Architecture Warehouse, in: Proceedings of the 7th European Conference on Software Architecture, ECSA'13, Springer-Verlag, Berlin, Heidelberg, 2013, pp. 146–161.

[31] N. Schuster, O. Zimmermann, C. Pautasso, ADkwik: Web 2.0 Collaboration System for Architectural Decision Engineering, in: Nineteenth International Conference on Software Engineering and Knowledge Engineering (SEKE), Knowledge Systems Institute Graduate School, 2007, pp. 255–260.

[32] D. Ameller, O. Collell, X. Franch, ArchiTech: Tool Support for NFR-guided Architectural Decision-Making, in: Requirements Engineering Conference (RE), 2012 20th IEEE International, 2012, pp. 315–316.

[33] T. Al-Naeem, I. Gorton, M. A. Babar, F. Rabhi, B. Benatallah, A Quality-driven Systematic Approach for Architecting Distributed Software Applications, in: 27th International Conference on Software Engineering, ICSE'05, ACM, New York, NY, USA, 2005, pp. 244–253.

[34] M. Shahin, P. Liang, Z. Li, Architectural Design Decision Visualization for Architecture Design: Preliminary Results of A Controlled Experiment, in: Proceedings of the 1st Workshop on Traceability, Dependencies and Software Architecture (TDSA), ACM, 2011, pp. 5–12.

[35] S. Herold, H. Klus, Y. Welsch, C. Deiters, A. Rausch, R. Reussner, K. Krogmann, H. Koziolek, R. Mirandola, B. Hummel, M. Meisinger, C. Pfaller, CoCoME - The Common Component Modeling Example, in: The Common Component Modeling Example: Comparing Software Component Models [result from the Dagstuhl research seminar for CoCoME, August 1-3, 2007], 2007, pp. 16–53.

[36] W. Bu, A. Tang, J. Han, An Analysis of Decision-centric Architectural Design Approaches, in: Proceedings of the 2009 ICSE Workshop on Sharing and Reusing Architectural Knowledge, SHARK'09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 33–40.

[37] R. Weinreich, I. Groher, A Fresh Look at Codification Approaches for SAKM: A Systematic Literature Review, in: Proceedings of the 8th European Conference on Software Architecture (ECSA), ECSA'14, Springer-Verlag, Berlin, Heidelberg, 2014, pp. 1–16.

[38] U. van Heesch, P. Avgeriou, A. Tang, Does decision documentation help junior designers rationalize their decisions? A comparative multiple-case study, Journal of Systems and Software 86 (6) (2013) 1545–1565.

[39] C. Zannier, F. Maurer, A qualitative empirical evaluation of design decisions, ACM SIGSOFT Software Engineering Notes 30 (4) (2005) 1–7.

[40] C. Miesbauer, R. Weinreich, Classification of Design Decisions: An Expert Survey in Practice, in: Proceedings of the 7th European Conference on Software Architecture, ECSA'13, Springer-Verlag, Berlin, Heidelberg, 2013, pp. 130–145.

[41] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, J. Rosenberg, Preliminary Guidelines for Empirical Research in Software Engineering, IEEE Trans. Softw. Eng. 28 (8) (2002) 721–734.

[42] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[43] R. C. Team, et al., R: A language and environment for statistical computing, R foundation for Statistical Computing.

[44] G. Vigderhous, The level of measurement and 'permissible' statistical analysis in social research, Pacific Sociological Review 20 (2) (1977) 61–72.

[45] H. R. Maurer, Todd J.; Pierce, A Comparison of Likert Scale and Traditional Measures of Self-Efficacy, Journal of Applied Psychology 83 (2) (1998) 324–329.

[46] S. S. Shapiro, M. B. Wilk, An analysis of variance test for normality (complete samples), Biometrika 3 (52).

[47] H. B. Mann, W. D. R., On a Test of Whether One of Two Random Variables is Stochastically Larger than the Other, Annals of Mathematical Statistics 18 (1) (1947) 50–60.

[48] M. Nowak, C. Pautasso, O. Zimmermann, Architectural Decision Modeling with Reuse: Challenges and Opportunities, in: Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge, SHARK'10, ACM, New York, NY, USA, 2010, pp. 13–20.

Paper I

Automatic Enforcement of Constraints in Real-time Collaborative Architectural Decision Making

The subsequent paper has been submitted and revised as follows:

P. Gaubatz, I. Lytra, and U. Zdun. Automatic Enforcement of Constraints in Real-time Collaborative Architectural Decision Making. *Journal of Systems and Software*, accepted for publication in January 2015.

This paper extends the concepts introduced in Paper H. More specifically, we observed the fact that existing tools and methods for collaborative architectural decision making focus mainly on sharing and reusing of knowledge, making trade-offs, and achieving consensus, but do not consider the various stakeholders' decision making constraints due to their roles in the development process. Therefore, we proposed a model-driven approach that provides means for making the decision making process – clearly a collaborative process – subject to different type of decision making constraints. Conceptually, these decision making constraints are quite similar to other types of access constraints that we have considered in our previous work (such as entailment constraints in Paper B or Paper G). Hence, we could extend CoCoADvISE with additional constraint enforcement capabilities in a similar way we did before in the case of CoCoForm (i.e., in Paper E, Paper F and Paper G). The evaluation of CoCoADvISE in a controlled experiment showed that our approach, besides preventing constraint violations, significantly increased both the time and effort related efficiency, as well as the effectiveness of users in collaborative decision making.

Automatic Enforcement of Constraints in Real-time Collaborative Architectural Decision Making

Patrick Gaubatz*, Ioanna Lytra, Uwe Zdun

Faculty of Computer Science, University of Vienna, Austria

Abstract

Making and documenting architectural design decisions becomes increasingly important in the process of software architecting. However, the remoteness of different decision stakeholders, ranging from local distribution in an office environment to globally distributed teams, as well as the different domain knowledge, expertise and responsibilities of the stakeholders hinder effective and efficient collaboration. Existing tools and methods for collaborative architectural decision making focus mainly on sharing and reusing of knowledge, making trade-offs, and achieving consensus, but do not consider the various stakeholders' decision making constraints due to their roles in the development process. To address this problem, we propose a meta-model for a set of decision making constraints, with precisely defined semantics, as well as a collaborative architectural decision making approach based on this meta-model. We also present tool support, called CoCoADvISE, which automatically enforces the constraints at runtime. The evaluation of this tool in a controlled experiment with 48 participants shows that our approach, besides preventing constraint violations, significantly increases both the time and effort related efficiency, as well as the effectiveness of users in collaborative decision making.

Keywords: Decision Making Constraint, Constraint Enforcement, Collaborative Decision Making, Architectural Decision Making, Reusable Architectural Decision Model, Controlled Experiment

1. Introduction

The trend of globally distributed projects in software and IT industries makes collaboration and coordination within dispersed teams challenging [11]. Unlike co-located project teams, geographically distributed partners need to overcome collaboration problems caused by the distance, different concerns of

*Corresponding author: Patrick Gaubatz, Faculty of Computer Science, University of Vienna, Währingerstraße 29, 1090 Vienna, Austria; Phone, +43-1-4277-785 20

Email addresses: patrick.gaubatz@univie.ac.at (Patrick Gaubatz), ioanna.lytra@univie.ac.at (Ioanna Lytra), uwe.zdun@univie.ac.at (Uwe Zdun)

stakeholders, and different development processes. While these challenges are particularly problematic in distributed project settings, even in a locally distributed environment, like offices on multiple different floors, efficient and effective collaboration is an issue. Software architecture can be seen as a tool for coordination in distributed software development, as a common understanding and agreement on issues at the architectural abstraction level can prevent coordination problems in later phases [34]. As architectural decisions have become a primary means for describing software architecture in recent years, the collaboration in architectural decision making and documenting should be supported. Farenhorst et al. point out the need of explicitly supporting collaboration between architects with appropriate tools and consider this aspect to be one of the five most important characteristics of software architecting [7].

Only a few of the existing tools for architectural decision management address collaboration in architectural decision making and documentation. Approaches that target collaborative architectural decision support (see, e.g., [31, 39, 53, 41, 20, 2, 4]) mainly focus on team building, achieving consensus, making of trade-offs, and sharing of architectural knowledge. However, none of these approaches considers the various stakeholders' decision making constraints due to their roles in the development process. This is despite the fact that such roles (see, e.g., [30, 45, 6]), their potentially diverging rights and duties and their potentially conflicting objectives (see, e.g., [36, 29, 34]) within a possibly constrained (see, e.g., [3, 18, 14]) decision making process actually do exist. As an exemplary decision making constraint, consider that a stakeholder with the role *Integration Architect* must be in agreement with a stakeholder with the role *Application Architect* before an architectural decision that concerns technical development aspects can be finalized.

In this paper, we therefore present a novel approach for augmenting reusable architectural decision models with such decision making constraints. Reusable decision models provide (similar to design patterns [10]) proven solutions – both application-generic and technology-specific – to various design issues along with their forces, consequences, and alternative solutions (see, e.g., [66, 67, 68, 65, 21]). We show how CoCoADvISE (Constrainable Collaborative Architectural Design Decision Support Framework), our prototype implementation, automatically enforces these decision making constraints at runtime. As an example, we describe how our approach can be applied in an industrial context, i.e., in the context of service-based platform integration.

So far there are only a few empirical studies on architectural decision making (see, e.g., [42, 59, 58]) in general or on the specific aspect of group decision making (see, e.g., [31, 37, 38]). As our work mainly deals with propositions about the efficiency and effectiveness of supporting automatic enforcement of constraints for humans, we decided to evaluate it using a controlled experiment with 48 participants. Our experiment provides evidence that automatic enforcement of decision making constraints significantly increases both the time and effort related efficiency and effectiveness of users of decision making tools.

The contributions of this paper are as follows:

- We propose automatic enforcement of decision making constraints, a new aspect to be considered in collaborative architectural decision making.
- We present CoCoADvISE, a constrainable collaborative decision making tool supporting automatic enforcement of constraints.
- We precisely specify a set of decision making constraints using first-order logic based on a formal meta-model (described in the appendix).
- We provide empirical evidence of the time and effort related efficiency and effectiveness of supporting automatic enforcement of constraints.

The remainder of the paper is structured as follows. We give an overview of the existing architectural decision management tools and discuss collaborative aspects and their challenges in Section 2. Section 3 motivates our work and provides a motivating example. In Section 4 we present our collaborative architectural decision making tool CoCoADvISE. Section 5 exemplifies the application of our approach in the context of service-based platform integration, which was investigated in the context of the EU research project INDENICA. The following Section 6 and 7 describe our experimental setting and the analysis of the results of the controlled experiment respectively. Section 8 discusses our findings, implications, as well as threats to validity, and Section 9 concludes.

2. Related Work

2.1. Architectural Design Decisions

Software architecture is seen more and more as a set of architectural decisions [13]. Capturing architectural design decisions is important for analyzing, understanding, and sharing the rationale and implications of these decisions and reducing the problem of architectural knowledge vaporization [10].

Approaches for capturing architectural decisions, using either templates [56], ontologies [19], or meta-models [66], concentrate on the reasoning on software architectures, capturing and reusing of architectural knowledge, as well as sharing and communicating of design decisions between stakeholders. In addition, patterns are regarded as proven knowledge for capturing architectural decisions and their rationale [10] and are considered often in the aforementioned approaches.

A substantial amount of work has been done in the direction of documenting architectural knowledge using architectural decision modeling (refer to [40] for a comparison of existing architectural decision models and tools). For instance, Jansen and Bosch propose a meta-model to capture decisions that consist of problems, solutions and their attributes [13]. Zimmermann et al.'s meta-model for capturing architectural decisions [69] consists of *Architectural Decisions (AD)* related to one or more *ADTopics* organized in *ADLevels*, entailing *ADAlternatives*, the selection of which leads to an *ADOutcome*. The advantage of such decision models is that they are reusable and can be used as guidance for architectural decision making activities, whenever recurring design issues emerge. Various reusable architectural decision models have been

documented in the literature, covering Service-oriented Architecture related solutions [67, 68], service-based platform integration [21], the design of domain specific languages [64], and model and metadata repositories [26]. To motivate and demonstrate our proposal we use such reusable architectural decision models as basis for decision making that involves different stakeholders.

2.2. Collaboration in Architectural Decision Making

Whereas several tools have been developed to ease capturing, managing and sharing of architectural decisions [40], only a few target explicitly the collaboration needs of distributed teams, i.e., when making architectural decisions in a group. The sharing of architectural knowledge is one of the main concerns of the architectural decision management tools AD_{kwik} [39], Knowledge Architect [20], and PAKME [2]. The collaboration in all cases is achieved by providing central repositories containing design pattern catalogs, documented architectural decisions, use case scenarios, and so on, accessible to all co-located or distributed software team members, without any automated support regarding the required collaborative work. In addition, Compendium [41] supports a visual environment for documenting and visualizing design rationale behind architectural design decisions for multiple users. In some cases, also, Wiki-based tools are proposed [4] to assist architectural knowledge management performed at geographically separate locations. However, collaborative work is not the main focus of these tools, and thus, the challenges of making and documenting architectural decisions collaboratively are not studied in aforementioned works.

Other recent proposals, such as Software Architecture Warehouse [31] and GADGeT [53], target the consensus making, the communication, and the progress tracking for group architectural decisions. Proposing, ranking, and voting for alternatives are main concepts that are integrated in the aforementioned tools. However, decision making constraints caused by different stakeholder roles, company policies or processes which may cause inconsistencies and additional effort are not considered. In our proposal, we extend the concepts of collaborative architectural decision making by introducing the automatic enforcement of such constraints during group decision making. Automatic support is thus an advantage of CoCoADvISE in comparison to the aforementioned tools, which do not target any automation in collaborative decision making.

Recent literature surveys and reviews that compare approaches and tools for architectural decision making and documentation (e.g., [40, 55, 38]) consider collaboration support as an important feature of these tools. In addition, according to a survey with 43 architects from industry conducted by Tofan et al., most of the architectural decisions are group decisions (86%) [54]. This finding is also validated in a different study by Miesbauer and Weinreich [27]. However, little empirical evidence – especially quantitative results – exists with focus on collaborative decision making by practitioners. Nowak and Pautasso have collected feedback from more than 50 focus groups of students regarding the usability and situational awareness support of their tool Software Architecture Warehouse [31]. Rekha et al. performed an exploratory study to investigate how

architectural design decisions are made in a group, what information is documented, and which tools are used by practitioners in the industry [37]. In our work, we target collaborative architectural decision making involving different stakeholder roles and various constraints.

2.3. Constraint Enforcement in the Context of Security and Access Control

Although constraint enforcement has not been considered in the context of architectural decision making yet, it is actually a quite well-studied topic in other contexts, such as security and access control. Especially access control in the context of business processes and workflows (see, e.g., [62, 15, 49]) introduces the notion of assigning (stakeholder) roles to each task in a process, which is similar to our notion of assigning a certain role to be responsible for a specific architectural decision. At runtime, each task/decision can only be performed/made by a user that owns the required role.

Task-based entailment constraints (see, e.g., [48, 50, 63]), which also originate from the context of workflows can, for instance, be used to enforce the four-eyes principle or other separation of duties constraints. In principle, such separation of duties constraints are similar in definition, checking, and enforcement to decision making constraints that require several stakeholders or roles to unanimously agree on a concrete solution for a specific architectural decision (which are, for instance defined in our approach). Unfortunately, to the best of our knowledge, there are no works that provide empirical evidence about the positive effects of constraint enforcement in these contexts.

2.4. Constraint Enforcement in the Context of Collaboration

In recent years, there has been a movement to embrace and facilitate collaboration in various software engineering tools. For example, numerous real-time collaborative Web-based Integrated Development Environments, such as Cloud9[1], Koding[2], Adinda [57] or Collabode [9] have been proposed. In addition, real-time collaborative Web-based modeling tools, such as Creatly[3] or Lucidchart[4], have also been proposed. Finally, more specialized software engineering tools, such as the collaborative Web-based software architecture evaluation tool presented by Maheshwari et al. [24], have emerged.

Similarly to these tools, our CoCoADvISE tool does not require its users to install or configure any software locally on their computers, which is one of the main benefits of Web applications. However, the aforementioned tools – unlike CoCoADvISE – consider different stakeholder roles, permissions and constraints. As the adoption of such tools in industrial contexts is likely to rise, this situation will probably change in the future [60]. Also, to the best of our knowledge, no comparable empirical studies on these tools and their underlying concepts have been performed.

[1] `http://c9.io`
[2] `http://koding.com`
[3] `http://creately.com`
[4] `http://lucidchart.com`

3. Motivation

3.1. Constrained Architectural Decision Making

Our approach provides tool support for architectural decision making constraints. Such constraints and the relationships and responsibilities of stakeholders that are formally expressed in those constraints are frequently discussed both in academic and industrial contexts. To motivate our approach, we summarize in this section some evidence from the literature.

Nord et al. present a structured approach for reviewing architecture documentation [30]. They provide an illustrative list of common stakeholder roles and their concerns in the decision making process. In particular, they also document the existence of potentially diverging rights and duties withing the decision making process, such as: "*For example, in safety-critical systems, the safety analyst is one of the most important stakeholders, as this person can often halt development on the spot if he or she believes the design has let a fault slip through.*". A study by Smolander et al. reveals and analyzes more than 20 stakeholder roles participating in architecture design in three different software companies, and concludes that further research on practices and tools for effective communication and collaboration among the varying stakeholders of the architecture design process are needed [45]. Eckstein documents experiences of a lead software architect on global agile projects [6]. This lead software architect propagates orchestration of the architect roles, i.e., having one lead architect and several subordinate architects, partitioned according to boundaries of subsystems, problem domains and sites. In the context of Enterprise Architecture (EA), an orthogonal study by van der Raadt et al. concludes that stakeholders pursue different, potentially conflicting, objectives, related to their specific role within the organization [36]. They also note that efficient collaboration between stakeholders is one of the main critical success factors for EA. Nakakawa et al. conducted a survey on effective execution of collaborative tasks during EA creation and highlighted – among others – the following challenges, reported by 70 enterprise architects [29]: (1) it is hard to reach consensus due to conflicting stakeholders' interests, (2) organization politics result in fuzzy decision making, (3) stakeholders are not accountable for their decisions, (4) lack of a clear decision making unit, (5) lack of a governance process that can ensure architecture compliance, (6) lack of supporting tools and techniques for executing collaborative tasks. Similar results have been reported by Päivi Ovaska et al. In [34] they conducted a case study in an international ICT company, analyzing coordination challenges in multi-site environment with geographically dispersed development teams. Most notably, they observed problems in coordination, such as lack of overall responsibility (i.e., unclear decision making) in architecture design.

In recent years, software development governance has been recognized as a key component for steering software development projects. In general, governance is about deciding the "*who, what, when, why, and how*" of (collaborative) decision making (see, e.g, [3, 5, 16]). Kofman et al. frame the ideal software development governance environment [18]: "*Every member of a team would know at any given point of time, what needs to be done, who is responsible for which*

task, and how to perform the tasks for which he or she is responsible". In the course of developing the IBM Software Development Governor prototype they also observed and documented the need for decision making constraints. That is, they mention decision making policies, such as framing the boundaries of the decision (i.e., who should participate, and when), or specifying how the decision is to be made (i.e., consensus or voting). Jensen et al. noted that "*there are many issues critical to governing software development, including decision rights, responsibilities, roles, accountability, policies and guidelines, and processes*" [14]. Interestingly, by studying NetBeans, an open source software project, they could actually identify similarities to these aforementioned governance issues. In summary, architectural decisions are key decisions, that can – in certain contexts – be subject to software development governance including the entailing decision making policies.

3.2. Motivating Example

To illustrate the concept of collaborative architectural decision making and to motivate the need for automatic enforcement of decision making constraints, let us consider a fictive online retailer company that wants to provide third-party online stores with a Web service for purchasing books and electronic gadgets. Based on functional requirements, as well as non-functional requirements, including requirements concerning secure and encrypted transactions, the company has to decide on what kind of Web service they are eventually going to deploy. For decisions that have such far-reaching implications throughout the system, company policies require that the involved software development teams, which are spread across several, geographically distributed offices, collaboratively participate in the decision making process.

In particular, the following decision making constraints can be derived from the company policies: First, all decision stakeholders with the role *Integration Architect* shall *unanimously* decide on the type of Web service to be exposed. *Application Architects* shall decide on technical details like a concrete transport protocol, and *Integration Architects* have to confirm and approve these decisions eventually. *Security Experts* shall propose a solution for the security and encryption related decisions. Finally, representatives from selected third-party companies shall be allowed to participate in the decision making process. However, their votes can be overruled at any time by internal stakeholders of the retailer company.

The company might now rely on guidelines that tell each stakeholder to exactly know their roles and duties within this decision making process and to stay compliant to these constraints in any decision process they participate in. However, with a growing number of stakeholders, stakeholder roles, responsibilities, duties and other forms of constraints, it becomes nearly impossible for a single participant to not (unintentionally) violate some of these policies (as will be shown in our controlled experiment in Section 6). Hence, we propose (in Section 4) a collaborative architectural decision making tool which automatically enforces such constraints.

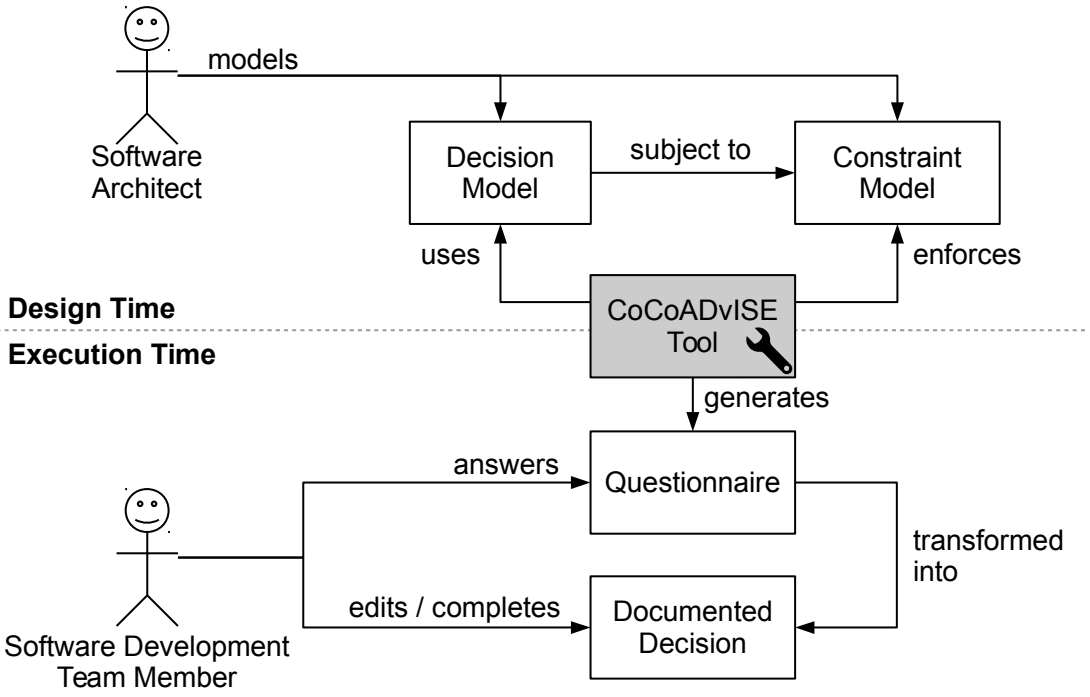


Figure 1: Overview of CoCoADvISE

4. CoCoADvISE Approach

CoCoADvISE[5] is a Web-based, collaborative tool that provides automated support for architectural decision making and documentation based on reusable decision models, as well as automatic enforcement of decision making constraints. It is based on previously presented work. More precisely, the tool is built upon the foundations of ADvISE (Architectural Design Decision Support Framework) [22] and CoCoForm (Constrainable Collaborative Form) [8], a real-time collaborative Web application framework. The main concepts of our approach are shown in Figure 1.

CoCoADvISE follows the reusable decision model approach (discussed already in Section 2.1) in which the documented reusable decisions can be instantiated as concrete decisions and thus used as guidance for architectural decision making activities, whenever recurring design issues emerge. The advantage of this approach is that the decisions must be created only once for a recurring design situation. In similar application contexts, a single decision model is reused multiple times for making multiple concrete decisions. A decision model can be (re-)used by transforming it into interactive questionnaires using model-driven techniques (see Section 4.1). Based on the outcomes of the questionnaires answered by Software Development Team Members, CoCoADvISE can automatically resolve potential constraints and dependencies (e.g., reveal follow-

[5]A demo of CoCoADvISE is available at: `https://piri.swa.univie.ac.at/cocoadvise`. Use the following user names to log in: experiment1 or experiment2 (no password required).

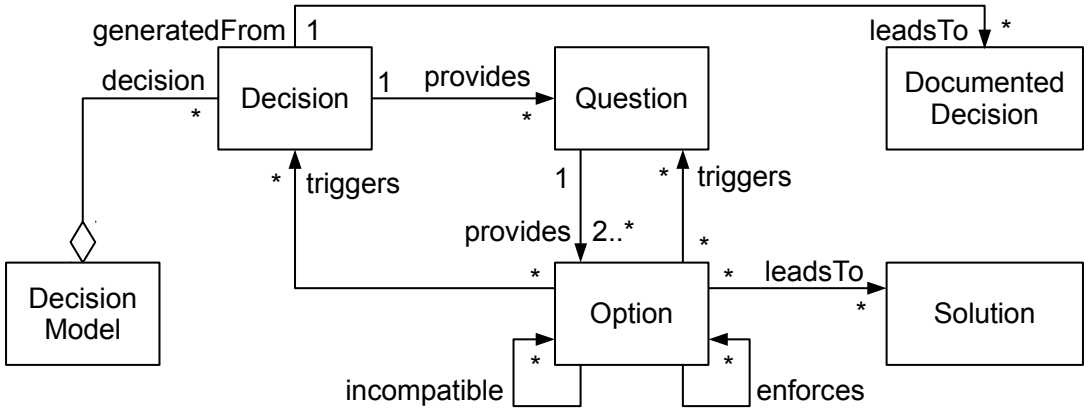


Figure 2: Conceptual Overview of CoCoADvISE's Decision Meta-model

on questions and decisions, deactivate options), recommend best-fitting design solutions, and eventually generate semi-complete architectural design decision (ADD) documentations (see Section 4.1).

The real-time collaboration features of CoCoADvISE enable multiple, possibly geographically dispersed software architects and stakeholders (i.e., Software Development Team Members) to participate in the group decision making and documentation process. In order to be applicable in industrial contexts, such as intra- and cross-organizational businesses, CoCoADvISE provides means for making this decision making process subject to constraints. It employs a model-driven approach in which reusable decision models are made subject to constraints, as can be seen in Figure 1 (see Section 4.2 and 4.3 for details).

The CoCoADvISE approach requires Software Architects to define abstract and reusable decision models at design time. If the decision making process has to be made subject to constraints, the Software Architects are supposed to formalize these requirements by defining constraint models and relating them to the corresponding decision models. Note that reusable decision models and constraints can only be defined and created using the ADvISE tool [22] while CoCoADvISE provides means for actually using these models.

4.1. Supporting Decision Making and Documentation

CoCoADvISE relies on reusable architectural decision models based on Questions, Options, and Criteria [23] that can be reused many times by the software architects in form of questionnaires, in order to guide architectural decision making. Such reusable architectural decision models are defined for recurring design situations, both domain and technology dependent and independent.

In particular, software architects define architectural decision models by creating instances of CoCoADvISE's decision meta-model at design time. While a detailed and formal representation of this meta-model can be found in Appendix A, Figure 2 provides a conceptual and graphical overview that may be more suitable for quickly grasping the core concepts and abstractions. Note that Figure 2, as well as the following Figure 3, 6, 7 and 8 represent UML2.2

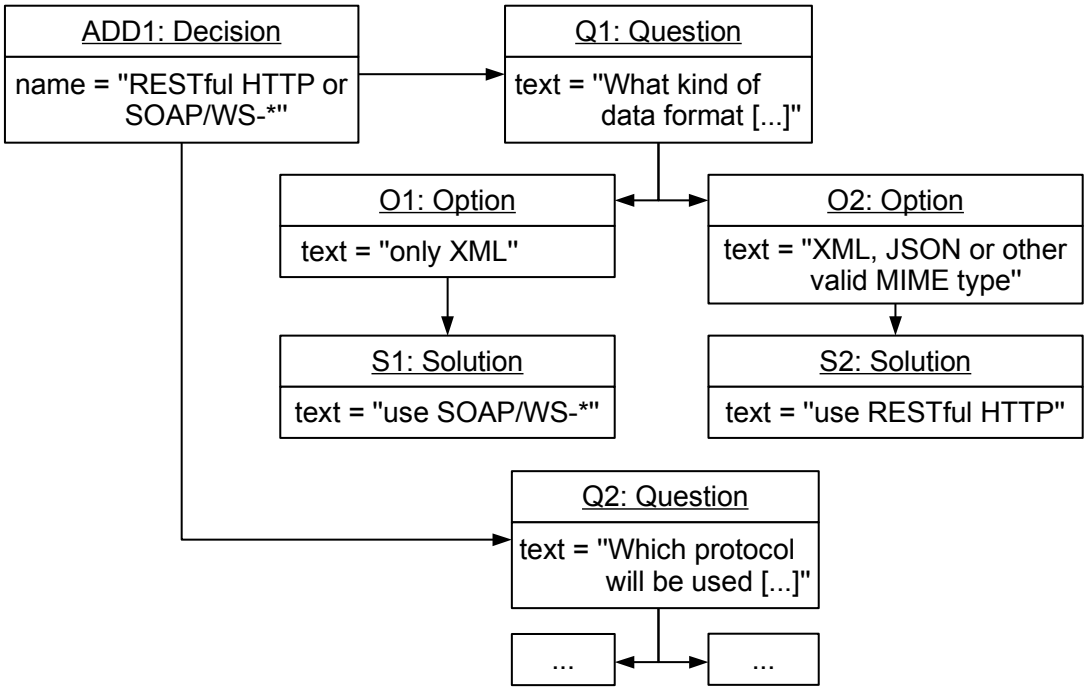


Figure 3: Exemplary Decision Model

Class and Object Diagrams with a minor syntactical deviation from the corresponding standard [33]. More precisely, we symbolize a relation's navigability with a filled arrowhead (i.e., ➙) instead of an open arrowhead.

As we can see, an architectural Decision Model consists of Decisions, Questions, Options, Solutions and relationships among them. Note that the usage of sans serif font indicates a reference to a class, e.g., Decision Model in Figure 2. These relationships allow for expressing consistency constraints and prescribing control flows like:

- If users choose Option x they must also choose Option y (enforces ➙).
- If users choose Option x they must not choose Option y (incompatible ➙).
- If users choose Option x they must also answer Question y (triggers ➙).
- If users choose Option x they must also decide on Decision y (triggers ➙).

The object diagram in Figure 3 depicts an excerpt of a very simplistic decision model that might be helpful in a situation where software architects have to decide on the kind of Web service to be deployed (as has been suggested in the motivating example from Section 3.2). This exemplary model consists of a single decision (i.e., ADD1) which has exactly one question (i.e., Q1). The question deals with the Web service's payload data format and can be answered by one of two options (i.e., O1 and O2). Finally, we can see, that each option leads to different solutions (i.e., S1 for O1 and S2 for O2). In general, Decisions

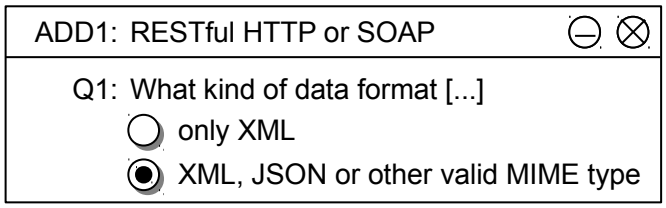


Figure 4: Exemplary Decision Questionnaire

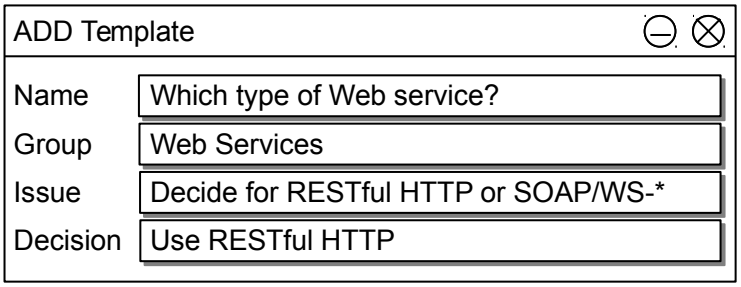
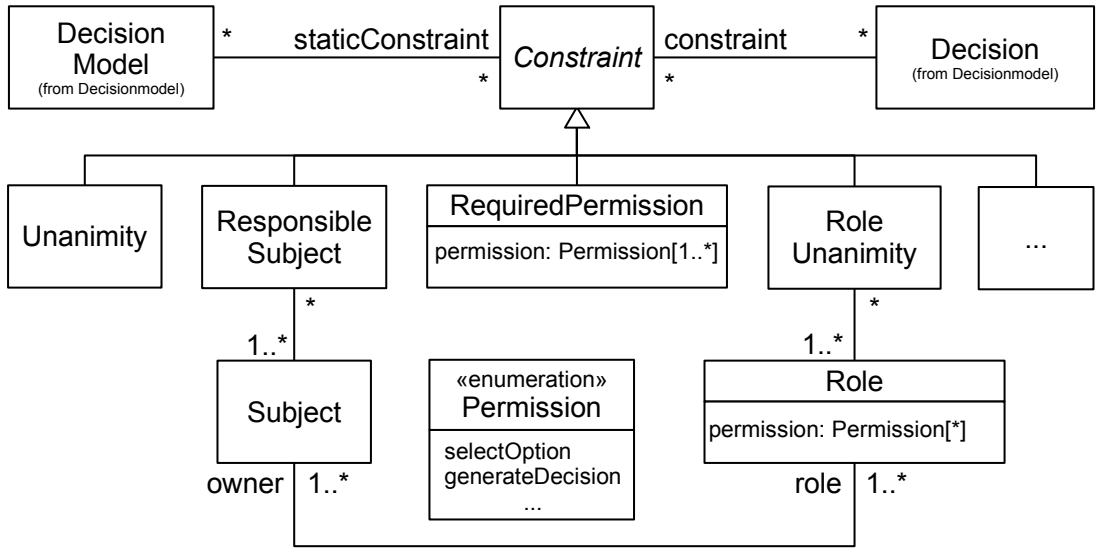


Figure 5: Exemplary Decision Template

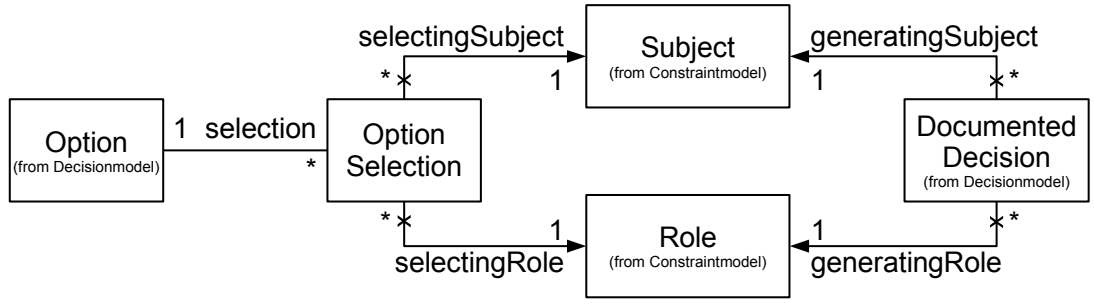
capture the essence of reusable architectural design decisions. Questions always belong to a particular superordinate Decision and are supposed to guide the software architect towards finding Solutions for the respective Decision. Finding Solutions involves choosing Options, thereby answering Questions.

In similar application contexts, software architects can reuse suitable architectural decision models. More precisely, reusing a decision model means instantiating the decision model and generating questionnaires which are used at execution time of the collaborative Web application by software development team members. As indicated in Figure 4 a questionnaire consists of questions and each question has to be answered by choosing exactly one option from a set of possible options. Thus, by answering these questionnaires, the best-fitting design solutions are recommended to the software development team members. CoCoADvISE takes the responsibility of verifying and guaranteeing the consistency of the decision model instances by automatically revealing follow-on questions and decisions, hiding, showing, enabling and/or disabling specific parts of the questionnaires in a way that users simply can not leave the questionnaire in an inconsistent state. For instance, whenever a user selects a specific option, the system automatically disables all other incompatible options.

Eventually, the selected options and gathered solutions can automatically be transformed into semi-completed architectural decision documentation templates similar to the one proposed by Tyree and Akerman [56]. For instance, if we consider the exemplary questionnaire depicted in Figure 4 and assume that a user has selected Option 2 (i.e., “XML, JSON or other valid MIME type”), we can transform this particular questionnaire into a decision documentation template (i.e., Documented Decision in Figure 1) like the one depicted in Figure 5. As we can see, many fields of these templates, such as “Name”, “Group”, “Issue”



(a) Constraint Model Extension



(b) Runtime Model Extension

Figure 6: Conceptual Overview of CoCoADvISE's Constrainable Decision Meta-model

and "Decision" can be completed (semi-)automatically by combining information that is encoded in the decision model (see, e.g., Figure 3) with user-provided input gathered via questionnaires (see, e.g., Figure 4).

4.2. Definition and Enforcement of Decision Making Constraints

In CoCoADvISE, reusable decision models are augmented with additional constraint elements. This section gives an overview of the generic meta-model for the specification of decision making constraints for reusable architectural decision models and details how these constraints are enforced at runtime.

The class diagrams depicted in Figure 6 provide a conceptual overview of the essential concepts of a Constrainable Decision Meta-model including Subjects, Roles, Permissions, and various types of Constraints. Technically, the Constrainable Decision Meta-model extends the previously presented Decision Meta-model (see Figure 2 and Section 4.1) with additional elements, relevant at design time (i.e., Figure 6(a)) and execution time (i.e., Figure 6(b)).

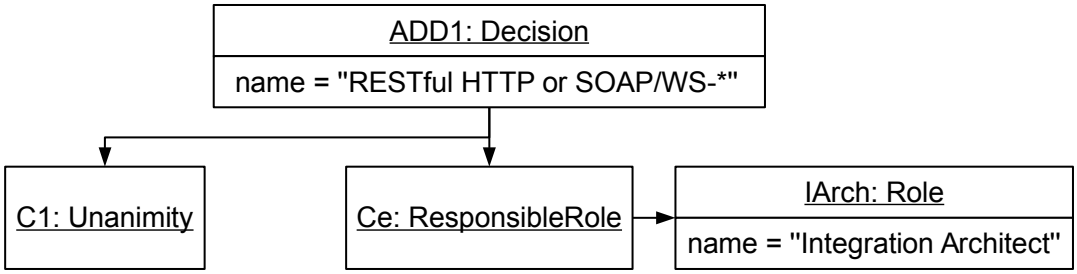


Figure 7: Exemplary Constraint Model

In our approach, Roles are used to model different job positions and scopes of duty within an organization. These Roles are equipped with the Permissions to perform certain tasks. Human users (i.e., Subjects) are assigned to Roles according to their work profile (see, e.g., [47]). This relation is a many-to-many relation, which means that a Subject can be assigned to numerous Roles and each Role can be owned by numerous Subjects.

CoCoADvISE provides two separate mechanisms for defining decision making constraints. First of all, we can make particular reusable architectural decisions subject to decision making constraints by assigning Constraints to Decisions. In addition, all reusable architectural decisions of a decision model can be made subject to decision making constraints at once by assigning the corresponding Constraints to the Decision Model, instead of a single decision. These two mechanisms are inherited by all subclasses of the abstract Constraint class (e.g., Unanimity or Responsible Subject).

As an example, let us revisit the motivating example from Section 3.2. Figure 7 depicts how the first set of decision making constraints can be implemented in CoCoADvISE. More precisely, we augment the previously defined exemplary decision model (see Figure 3) by attaching decision making constraints. By defining both a Responsible Role and a Unanimity constraint we can formalize the requirement that stakeholders with the role *Integration Architect* have to *unanimously* decide on the type of Web service. Note that we have to parametrize the Responsible Role constraint by assigning the Role object, which represents the *Integration Architect* role, to it. On the contrary, a Unanimity constraint can and does not need any further parametrization. According to Figure 6(a) the three additional classes that are used for parametrization of some constraints are: Subject, Role, and Permission. A Subject represents a person, i.e., a user/stakeholder of the system (see, e.g., [49]). Roles are assigned to subjects, and through their roles the subjects are granted Permissions. A Permission models a right to perform a certain action within the system. In CoCoADvISE we mainly focus on constraining two fundamental actions, i.e., selecting options and generating decisions. Figure 6 reflects this circumstance by explicitly listing the corresponding permissions selectOption and generateDecision. Other permissions could be added to the tool as extensions.

Figure 6(b) shows how we extend our original decision meta-model of Figure 2 with additional runtime-specific elements and relations. For the purpose of

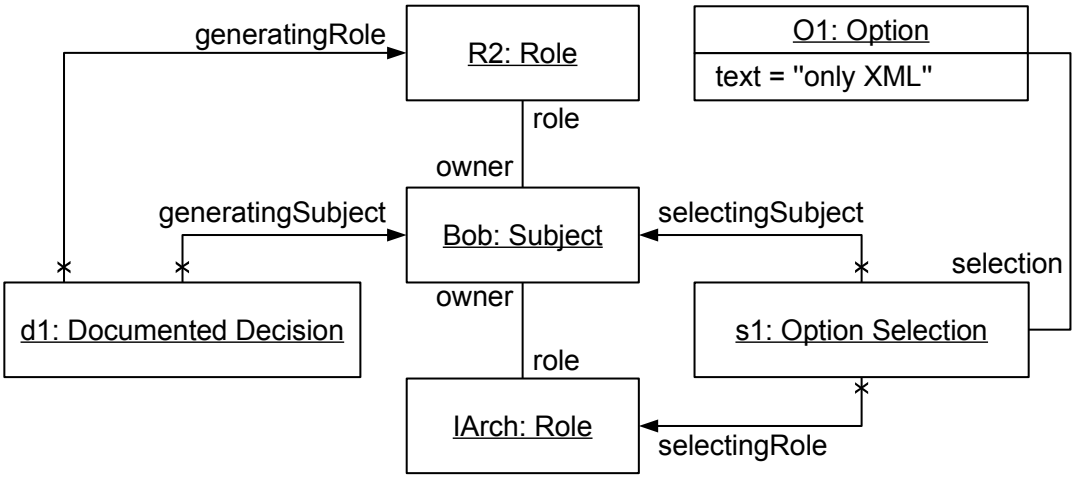


Figure 8: Exemplary Constraint Runtime Model

enforcing certain decision making constraints we have to introduce – among others – the concept of an Option Selection. At execution time, the existence of an Option Selection object means that the corresponding Option has been selected by a user (i.e., executing a selectOption action). More precisely, the relations selectingSubject and selectingRole of this Option Selection object are supposed to point to the user's corresponding Subject respective Role objects. Analogously, the generatingSubject and generatingRole relations of a Documented Decision object are supposed to refer to the Subject and Role objects of the user who has generated a decision (i.e., executing a generateDecision action). Figure 8 exemplifies these runtime-specific elements and relations. In this particular example, there is one Subject (i.e., Bob), that owns two different Roles (i.e., IArch and R2). Assuming, that Bob selects Option O1 at runtime, Option Selection s1 is newly instantiated. In addition, the selectingSubject and selectingRole relations of s1 are set to Bob and IArch (hereby assuming that IArch is the Role that Bob is currently using). In summary, s1 captures the fact that Bob has selected O1 using the Role IArch. Later, Bob generates a Documented Decision d1 using another Role, i.e., R2. As a result, the generatingSubject and generatingRole relations of d1 are set to Bob and R2 to capture the Subject and Role responsible for this event.

4.3. Logical Specification of Decision Making Constraints

In this section, we present the precise definition of the decision making constraints using first-order logic as a formalism that is abstract and technology-independent but can still easily be mapped to different existing constraint languages used in modeling and software development, such as OCL[6], the Check

[6] `http://www.omg.org/spec/OCL/`

language of the Eclipse M2T project[7], or Frag's FCL constraints[8]. In order to enable this precise definition, we needed to map the meta-models from Figures 2 and 6 to first-order logic as well. For the sake of completeness we provide this formalization in Appendix A. The reader can refer to Appendix A for a complete reference of the semantics of the constraint meta-model but we note that this is not necessary for understanding the constraint types and their application. In particular, Definition A.1 provides a list of elements and their relations, Definition A.2 presents crucial model invariants to be considered at design time, and Definition A.3 lists model invariants relevant at execution time.

The following list of Constraint Type (CT) Definitions constitutes our proposed set of decision making constraints. Each Constraint Type Definition consists of a narrative description, a formal definition of model invariants, expressed using first-order logic (see, e.g., [46]), as well as an exemplary application of the respective constraint type. A constrainable decision model that fulfills the invariants of a Constraint Type Definition CT_n is said to be compliant with CT_n. Note that this list is *not* an exhaustive list of all possible constraint types. Instead, we selected a set of constraint types that we believe to be representative and common in real-life scenarios.

The following three Constraint Types (CT_{1-3}) belong to the family of **Unanimity** constraints. Such constraints can be used to enforce consensus finding among a particular set of stakeholders. As has been mentioned in Section 4.2, CoCoADvISE focuses on constraining the process of selecting options and generating decisions. In connection with this, Unanimity constraints concern the option selection process.

CT_1 **(General) Unanimity:** Each question of a decision that is subject to a (general) unanimity constraint has to be answered unanimously. More precisely, a question is said to be answered unanimously if all subjects have agreed on the concrete set of options to select. This is the case when all subjects (i.e., "users") have selected the same set of options.

In order to be able to express this model invariant, we need to define the following required mappings first:

$cd(c) = \{d_1, ..., d_n\}$	Set of decisions $\{d_1, ..., d_n\}$ that is constrained by constraint c (see Definition A.1.19)
$qda(d) = \{q_1, ..., q_n\}$	Set of questions $\{q_1, ..., q_n\}$ that belong to decision d (see Definition A.1.2)
$oqa(q) = \{o_1, ..., o_n\}$	Set of options $\{o_1, ..., o_n\}$ that belong to question q (see Definition A.1.3)
$soa(o) = \{os_1, ..., os_n\}$	Set of (option) selections $\{os_1, ..., os_n\}$ that belong to option o (see Definition A.1.9)

In addition, the following set needs to be defined (see Definition A.1):

[7] `https://www.eclipse.org/modeling/m2t/`
[8] `http://frag.sourceforge.net/`

C_U	An element of C_U is called *Unanimity Constraint*
D	An element of D is called *Decision*
Q	An element of Q is called *Question*
O	An element of O is called *Option*
O_S	An element of O_S is called *Option Selection*

Therefore:

For each unanimity constrained question (q), if one option (o_x) has been selected ($soa(o_x) \neq \emptyset$), all other options ($o_y \neq o_x$) must not be selected ($soa(o_y) = \emptyset$).

$$\forall c \in C_U(\forall d \in cd(c)(\forall q \in qda(d)(\forall o_x \in oqa(q)(\forall o_y \in oqa(q)(\\ o_x \neq o_y \wedge soa(o_x) \neq \emptyset \Rightarrow soa(o_y) = \emptyset)))))$$

Example: Suppose there is a decision ($d_1 \in D$), a question ($q_1 \in Q$), two options ($\{o_1, o_2\} \subseteq O$), a unanimity constraint ($c_1 \in C_U$) and two option selections ($\{os_1, os_2\} \subseteq O_S$). The question belongs to the decision ($qda(d_1) = \{q_1\}$) and the options belong to the question ($oqa(q_1) = \{o_1, o_2\}$). Finally, the decision is made subject to the unanimity constraint ($cda(d_1) = \{c_1\}$). If we assume that option o_1 has been selected twice by different "users" ($soa(o_1) = \{os_1, os_2\}$), then o_2 could not have been selected ($soa(o_2) = \emptyset$) and decision d_1 is said to be compliant regarding the unanimity constraint c_1. Hereby, an option selection such as os_1 (and os_2 respectively) maps to exactly one subject and one role (see Definition A.1.10, Definition A.1.11 and Figure 6(b)). In other words, an option selection links a particular option with a subject/role combination (i.e., a "user"). Conversely, if both o_1 and o_2 have been selected once ($soa(o_1) = \{os_1\}$ and $soa(o_2) = \{os_2\}$), the invariant would not be fulfilled and the unanimity constraint c_1 is said to be violated.

CT_2 **Role Unanimity:** Contrary to an ordinary unanimity constraint, the role unanimity constraint allows for precisely specifying the set of subjects that are supposed to unanimously agree on a decision. More specifically, it allows for providing a specific set of roles. All subjects that own at least one of these roles have to unanimously agree on the same set of options for the questions of a decision.

A role unanimity constraint may be used to enforce the exemplary requirement "stakeholders with the role *Integration Architect* shall *unanimously* decide on the type of Web service" from the motivating example in Section 3.2.

Required mappings:

$sr(os) = r$	Role r that has been used to perform selection os (see Definition A.1.11)
$rruca(c) = \{r_1, ..., r_n\}$	Set of roles $\{r_1, ..., r_n\}$ that must unanimously agree on decisions that are constrained by constraint c (see Definition A.1.14)

Required sets:

C_{U_R}	An element of C_{U_R} is called *Role Unanimity Constraint*
R	An element of R is called *Role*

Therefore:

For each role unanimity constrained question (q), if one option (o_x) has been selected ($soa(o_x) \neq \emptyset$) by one of the specified roles ($sr(o_x) \in rruca(c)$), all other options must not be selected ($soa(o_y) = \emptyset$).

$$\forall c \in C_{U_R}(\forall d \in cd(c)(\forall q \in qda(d)(\forall o_x \in oqa(q)(\forall o_y \in oqa(q)(\\ o_x \neq o_y \wedge soa(o_x) \neq \emptyset \Rightarrow soa(o_y) = \emptyset \wedge \forall os_x \in soa(o_x)(\\ sr(os_x) \in rruca(c)))))$$

Example: Based on the example for CT_1 (i.e., d_1, q_1, o_1, o_2, os_1 and os_2), we assume that the decision (d_1) is made subject to a role unanimity constraint ($c_1 \in C_{U_R}$ and $cda(d_1) = \{c_1\}$). The constraint requires that subjects with a particular role ($r_1 \in R$ and $r_1 \in rruca(c_1)$) have to unanimously agree on a particular option. If we assume that option o_1 has been selected twice by different "users" ($soa(o_1) = \{os_1, os_2\}$) with that particular role ($sr(os_1) = sr(os_2) = r_1$), then o_2 could not have been selected ($soa(o_2) = \emptyset$) and decision d_1 is said to be compliant regarding the role unanimity constraint c_1. Conversely, if both o_1 and o_2 have been selected once ($soa(o_1) = \{os_1\}$ and $soa(o_2) = \{os_2\}$), the invariant would not be fulfilled and the role unanimity constraint c_1 is said to be violated. The constraint is also violated if either option has been selected by a role other than the required one ($sr(os_1) \notin \{r_1\}$ or $sr(os_2) \notin \{r_1\}$).

CT_3 **Subject Unanimity:** The subject unanimity constraint is similar to the role unanimity constraint, but instead of specifying a set of roles (supposed to unanimously agree on a decision), it allows for directly specifying a set of subjects.

Required mappings:

$ss(os) = s$	Subject s that has performed selection os (see Definition A.1.10)
$ssuca(c) = \{s_1, ..., s_n\}$	Set of subjects $\{s_1, ..., s_n\}$ that must unanimously agree on decisions that are constrained by constraint c (see Definition A.1.15)

Required sets:

C_{U_S}	An element of C_{U_S} is called *Subject Unanimity Constraint*
S	An element of S is called *Subject*

Therefore:

For each subject unanimity constrained question (q), if one option (o_x) has been selected ($soa(o_x) \neq \emptyset$) by one of the specified subjects ($ss(o_x) \in$

ssuca(c)), all other options must not be selected ($soa(o_y) = \emptyset$).

$$\forall c \in C_{U_S}(\forall d \in cd(c)(\forall q \in qda(d)(\forall o_x \in oqa(q)(\forall o_y \in oqa(q)(o_x \neq o_y \wedge soa(o_x) \neq \emptyset \Rightarrow soa(o_y) = \emptyset \wedge \forall os_x \in soa(o_x)(ss(os_x) \in ssuca(c)))))$$

Example: Based on the example for CT_1 (i.e., d_1, q_1, o_1, o_2, os_1 and os_2), we assume that the decision (d_1) is made subject to a subject unanimity constraint ($c_1 \in C_{U_S}$ and $cda(d_1) = \{c_1\}$). The constraint requires that particular subjects ($\{s_1, s_2\} \subseteq S$ and $\{s_1, s_2\} \subseteq ssuca(c_1)$) have to unanimously agree on a particular option. If we assume that option o_1 has been selected twice ($soa(o_1) = \{os_1, os_2\}$) by that particular subjects ($ss(os_1) \in \{s_1, s_2\}$ and $ss(os_2) \in \{s_1, s_2\}$), then o_2 could not have been selected ($soa(o_2) = \emptyset$) and decision d_1 is said to be compliant regarding the subject unanimity constraint c_1. Conversely, if both o_1 and o_2 have been selected once ($soa(o_1) = \{os_1\}$ and $soa(o_2) = \{os_2\}$), the invariant would not be fulfilled and the subject unanimity constraint c_1 is said to be violated. The constraint is also violated if either option has been selected by a subject other than the required ones ($ss(os_1) \notin \{s_1, s_2\}$ or $ss(os_2) \notin \{s_1, s_2\}$).

The following two Constraint Types (CT_{4-5}) belong to the family of **Responsibility** constraints. Such Responsibility constraints concern the decision generation process and can be used to precisely specify which particular set of stakeholders shall be entitled to eventually make (i.e., generate) a decision.

CT_4 **Responsible Role:** Each decision that is subject to a responsible role constraint shall be transformable into documented decisions *only* by subjects that own at least one of given set of roles. In other words, this constraint allows for preventing subjects that do not own specific roles from making (i.e., generating) certain decisions.

A responsible role constraint may be used to enforce the exemplary requirement "*Security Experts* shall propose a solution for the security and encryption related decisions" from the motivating example in Section 3.2.

Required mappings:

$dda(d) = \{dd_1, ..., dd_n\}$	Set of documented decisions $\{dd_1, ..., dd_n\}$ that have been generated from decision d (see Definition A.1.6)
$gr(dd) = r$	Role r that has been used to generate documented decision dd (see Definition A.1.8)
$rrrca(c) = \{r_1, ..., r_n\}$	Set of roles $\{r_1, ..., r_n\}$ that are responsible for generating decisions that are constrained by constraint c (see Definition A.1.16)

Required sets:

C_{R_R}	An element of C_{R_R} is called *Responsible Role Constraint*

Therefore:

If documented decisions (dd) have been generated from a responsible role constrained decision (d), these decisions have to be generated by a subject using one of the specified (responsible) roles ($gr(dd) \in rrrca(c)$).

$$\forall c \in C_{R_R}(\forall d \in cd(c)(\forall dd \in dda(d)(gr(dd) \in rrrca(c))))$$

Example: Based on the example for CT_1 (i.e., d_1, q_1, o_1, o_2, os_1 and os_2), we assume that the decision (d_1) is made subject to a responsible role constraint ($c_1 \in C_{R_R}$ and $cda(d_1) = \{c_1\}$). The constraint requires that subjects with a particular role ($r_1 \in R$ and $r_1 \in rrrca(c_1)$) have to generate decisions that are based on d_1. If we assume that decision dd_1 has been generated from d_1 ($dd_1 \in dda(d_1)$) by a subject with that particular role ($gr(dd_1) = r_1$), then decision d_1 is said to be compliant regarding the responsible role constraint c_1. Conversely, if dd_1 had been generated by a role other than the required one ($gr(dd_1) \notin \{r_1\}$), the invariant would not be fulfilled and the responsible role constraint c_1 is said to be violated.

CT_5 **Responsible Subject:** Similarly to the responsible role constraint, the responsible subject constraint allows for precisely specifying a set of subjects that are supposed to make (i.e., generate) certain decisions. Conversely, all other subjects shall not be allowed to do so.

Required mappings:

$gs(dd) = s$	Subject s that has generated documented decision dd (see Definition A.1.7)
$srsca(c) = \{s_1, ..., s_n\}$	Set of subjects $\{s_1, ..., s_n\}$ that are responsible for generating decisions that are constrained by constraint c (see Definition A.1.17)

Required sets:

C_{R_S}	An element of C_{R_S} is called *Responsible Subject Constraint*

Therefore:

If documented decisions (dd) have been generated from a responsible subject constrained decision (d), these decisions have to be generated by one of the specified (responsible) subjects ($gs(dd) \in srsca(c)$).

$$\forall c \in C_{R_S}(\forall d \in cd(c)(\forall dd \in dda(d)(gs(dd) \in srsca(c))))$$

Example: Based on the example for CT_1 (i.e., d_1, q_1, o_1, o_2, os_1 and os_2), we assume that the decision (d_1) is made subject to a responsible subject constraint ($c_1 \in C_{R_S}$ and $cda(d_1) = \{c_1\}$). The constraint requires that a particular subject ($s_1 \in S$ and $s_1 \in srsca(c_1)$) has to generate decisions that are based on d_1. If we assume that decision dd_1 has been

generated from d_1 ($dd_1 \in dda(d_1)$) by that particular subject ($gs(dd_1) = s_1$), then decision d_1 is said to be compliant regarding the responsible subject constraint c_1. Conversely, if dd_1 had been generated by a subject other than the required one ($gs(dd_1) \notin \{s_1\}$), the invariant would not be fulfilled and the responsible subject constraint c_1 is said to be violated.

The following Constraint Type (CT_6) can be considered a generic **Access Control** constraint. In the context of CoCoADvISE it concerns constraining the set of stakeholders that shall be *authorized* to select options or generate decisions. Whereas CT_6 concerns authorization (i.e., who shall potentially be entitled to do what), CT_{2-5} concerns *obligation* (i.e., who must do what).

CT_6 **Required Permission:** A required permission constrained decision respective decision model enforces that subjects that do not own certain permissions (i.e., via their roles) within the system must not perform the corresponding actions.

Required mappings:

$prpca(c) = \{p_1, ..., p_n\}$	Set of permissions $\{p_1, ..., p_n\}$ that are required for performing actions concerning decisions that are constrained by constraint c (see Definition A.1.18)
$pra^{-1}(r) = \{p_1, ..., p_n\}$	Set of permissions $\{p_1, ..., p_n\}$ that are owned by role r (see Definition A.1.5)

Required sets:

C_P	An element of C_P is called *Required Permission Constraint*

Therefore:

If documented decisions (dd) have been generated from a decision (d) that is subject to a required permission constraint (c), the generating role ($gr(dd)$) must own the corresponding permission ($generateDecision \in pra^{-1}(gr(dd))$) to generate decisions.

$$\forall c \in C_P(generateDecision \in prpca(c) \Rightarrow \forall d \in cd(c)(\forall dd \in dda(d)(generateDecision \in pra^{-1}(gr(dd)))))$$

If options of a decision (d) that is subject to a required permission constraint (c) have been selected (os), the selecting role ($sr(os)$) must own the permission to select options ($selectOption \in pra^{-1}(sr(os))$).

$$\forall c \in C_P(selectOption \in prpca(c) \Rightarrow \forall d \in cd(c)(\forall q \in qda(d)(\forall o \in oqa(q)(\forall os \in soa(o)(selectOption \in pra^{-1}(sr(os)))))))$$

Note that similar invariants for other actions, such as deleting questionnaires or decisions, have been omitted for brevity.
Example: Based on the example for CT_1 (i.e., d_1, q_1, o_1, o_2, os_1 and os_2),

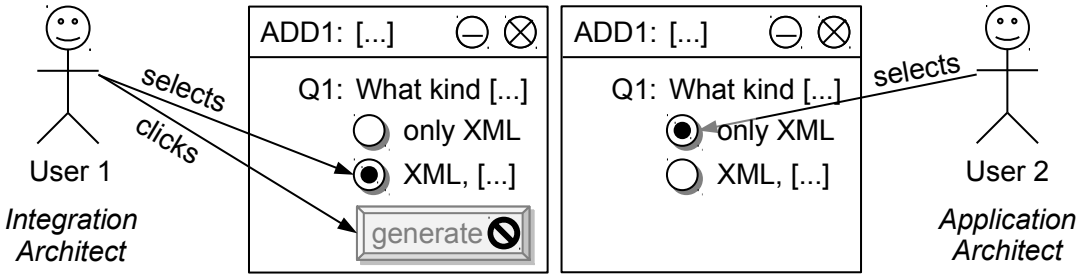


Figure 9: Enforcing Exemplary Constraints at Execution Time

we assume that the decision model is made subject to a required permission constraint ($c_1 \in C_P$). The constraint mandates – among other things – that all option selections have to be performed by subjects owning the required *selectOption* role. If we assume that option selection os_1 has been performed by a subject using a role ($r_1 \in R$) that owns that particular permission ($selectOption \in pra^{-1}(r_1)$), then the decision model is said to be compliant regarding the required permission constraint c_1. Conversely, if r_1 did not own that particular permission ($selectOption \notin pra^{-1}(r_1)$), the invariant would not be fulfilled and the required permission constraint c_1 is said to be violated.

At execution time, CoCoADvISE interprets constrainable decision models and constantly checks the system's compliance to the defined constraints while the software development team members are making and documenting decisions. The actual constraint enforcement logic is mainly embedded in the user interface of the collaborative Web application. The constraints are automatically enforced in the same way, the system also guarantees the consistency of the decision model instances, i.e., by automatically hiding, showing, disabling, etc. specific parts of the questionnaires and the user interfaces in a way that users simply can not violate any defined constraints at all. Figure 9 visualizes a possible runtime situation of the previously defined decision (see Figure 3) and constraint model (see Figure 7). We can see that the specified responsible role constraint is enforced by showing the "generate" button only to those users that own the required role (i.e., User 1, which owns the role *Integration Architect*), while hiding it for all other users (i.e., User 2). Given that User 1 chooses option 2 and User 2 chooses another option 1, the system correctly enforces the unanimity constraint by disabling the "generate" button. As soon as all users have unanimously agreed on the same set of options, the system will eventually enable the button again, allowing User 1 to generate a documented decision.

4.4. Implementation Details

CoCoADvISE is a real-time collaborative Single-page Web application that is founded on Google's Web application framework AngularJS[9]. Thus, it is exe-

[9] `http://angularjs.org`

cuted mostly client-side (i.e., in the user's Web browser). The model invariants for each constraint type (i.e., CT_{1-6}) have been transformed manually into client-side executable code. A key component of real-time collaborative Web applications is a real-time model synchronization engine that allows for synchronizing the shared application state with all clients. CoCoADvISE leverages Racer[10] for synchronizing the decision making process with all clients. Racer consists of both client-side and server-side executed code. All server-side code is executed in Node.js[11], an asynchronous event driven framework and runtime environment based on Google's V8[12] JavaScript engine. The back-end of this Thin Server Architecture persists the application state using MongoDB[13], a document-oriented (i.e., NoSQL) database. According to the tool CLOC[14] (Count Lines of Code), the application consists of nearly 2100 lines of client-side executed JavaScript code, roughly 1000 lines of HTML code and 150 lines of server-side executed Javascript code.

In addition to collaborative editing of questionnaires and architectural design decisions, CoCoADvISE also provides a simple chat, which allows all stakeholders to participate in discussions concerning the decision making process.

4.5. Motivating Example Resolved

In the course of revisiting the motivating example from Section 3.2, Figure 10 shows screenshots of CoCoADvISE, currently displaying the mentioned decision model excerpt from an *Integration Architect's* point of view. ① indicates that another *Software Architect* with the name "experiment1" has currently selected a different option than "we". Due to the Unanimity constraint, the system automatically disables the corresponding "Generate decision" button ②. By selecting the same option as "experiment1" ③ "we" can successfully resolve this constraint violation and eventually the system allows for generating the corresponding documented decision ④. Finally, ⑤ displays an excerpt of this generated decision.

5. Application of Constrainable Collaboration in Service-based Platform Integration

In practice, more than two roles with various intertwining responsibilities, rights, and permissions are involved in decision making on architectural-related issues. In this section, we present the application of constrainable collaborative architectural decision making in the domain of service-based platform integration. In particular, we discuss the implementation of our approach to support architecture governance in tailoring of heterogeneous domain-specific platforms,

[10] `http://github.com/codeparty/racer`
[11] `http://nodejs.org`
[12] `http://code.google.com/p/v8`
[13] `http://mongodb.org`
[14] `http://cloc.sourceforge.net`

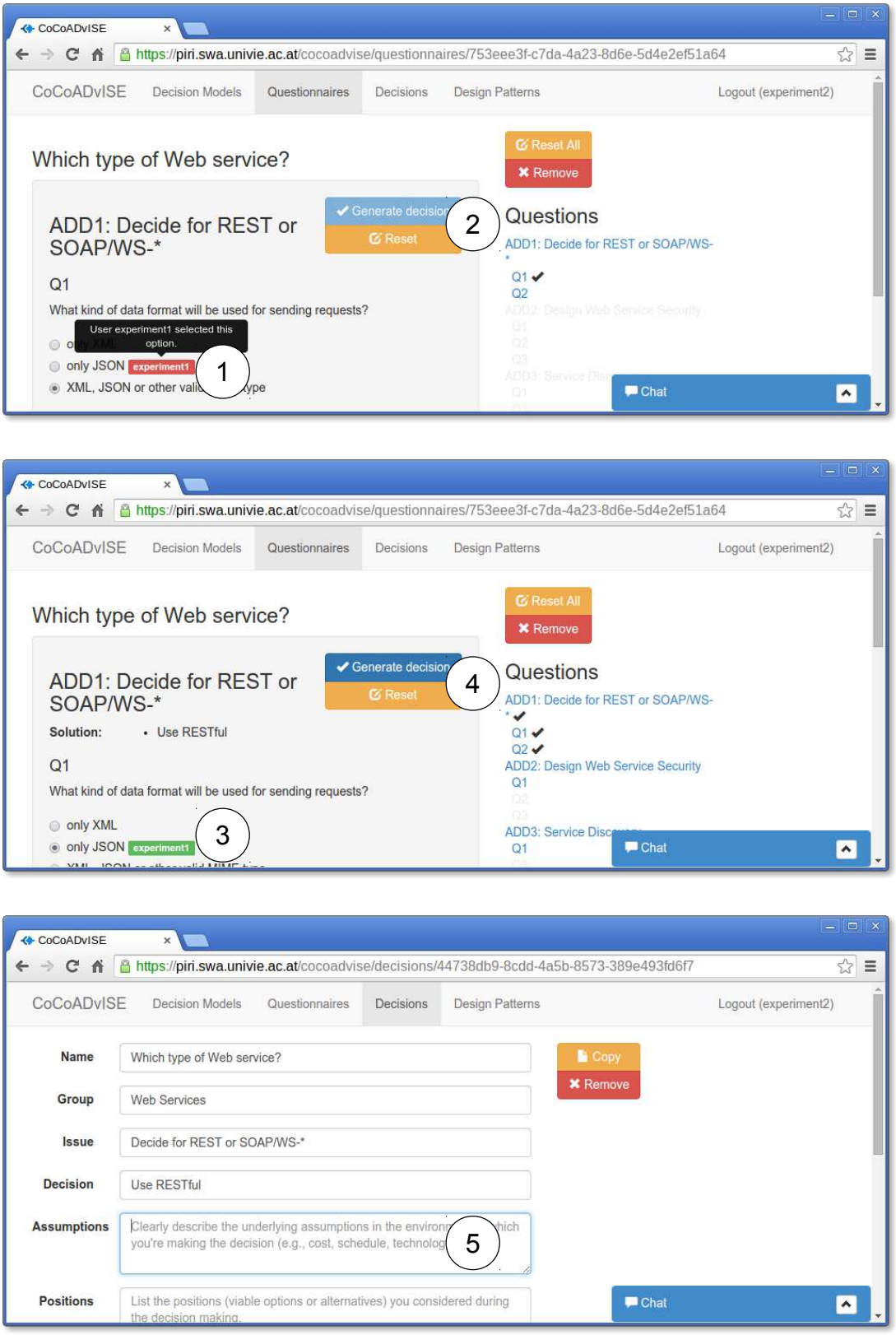


Figure 10: Screenshots of CoCoADvISE

which was investigated in the context of the EU research project INDENICA[15], which included project partners from the industry. INDENICA and in particular one of its technical reports [44] on role-based governance originally inspired us to devising the approach presented in this paper.

Building domain-specific service platforms is necessary for fulfilling specific requirements of various domains – sometimes incompatible to each other – in order to ease the development of services and applications within the domain. Such domain-specific service platforms may form a family of platforms, in which member platforms share assets. In this way, the INDENICA approach tailors the platforms towards the application domains and provides methods and tools for designing and implementing a Virtual Domain-Specific Service Platform (VSP). As described in the technical report [44], this approach requires the implementation and application environment to be considered from an organizational and human behavior point of view. This can be addressed by introducing different types of governance, such as Corporate Governance, Business Process Management Governance (BPM Governance), Information Technology Governance (IT Governance), Enterprise Architecture Governance (EA Governance), SOA Governance, and Architecture Governance in the different software processes. We will focus on Architecture Governance which concentrates on system and platform architecture related aspects.

Architecture Governance is defined as *"the practice and orientation by which enterprise architectures and other architectures are managed and controlled at an enterprise-wide level"* [52]. To develop a consistent architecture and ensure the evolution, adaptation, and modification of integrating domain-specific service platforms in INDENICA, architecture governance is of key importance. That is, without such governance the risk of e.g., wrong usage of services, project failure, over-complex applications, and design erosion increases. During design time, the following key participating roles have been defined:

Platform Provider is a technology expert and describes the current variability and the variability binding process of the existing platform he owns.

Platform Variant Creator is responsible for binding unresolved variability in base platform(s) and for creating an executable platform variant.

Platform Architect is responsible for VSP requirements, variability within VSP, baseline architecture and adaptation behavior of VSP.

Platform Integrator generates the VSP instance.

In Table 1, we list 16 decision categories related to service-based platform integration design that have been described in [44]. These decision categories include decision points that need to be considered, discussed, and eventually, resolved by the various stakeholders in collaboration. For instance, for Decision

[15] `http://www.indenica.eu`

Decision Category	Constraints	
	C_{R_R}	C_{U_R}
01. Decide variability modeling (describing the current variability and the variability binding process of the base platform).	PP	PP
02. Implement base platform relevant requests.	PP	
03. Bind unresolved variability in base platform.	PVC	
04. Create an executable platform variant optional.	PVC	
05. Decide additional functionality not covered by the base platform.	PVC	PP, PVC
06. Decide variability modeling of additional functionality.	PVC	PP, PVC
07. Implement domain platform relevant change requests.	PVC	PP, PVC
08. Design the VSP Capabilities (requirements management).	PA	
09. Decide the variability within VSP.	PA	
10. Decide the VSP constraints.	PA	
11. Decide the VSP orchestration.	PA	
12. Create the baseline architecture.	PA	PA, PI
13. Create the baseline adaptation behavior of VSP.	PA	PA, PI
14. Decide/Implement monitoring/adaptation rules.	PI	PA, PI
15. Decide integration of appropriate platforms.	PI	PA, PI
16. Generate the integration of the domain platforms to the VSP.	PI	

PP: Platform Provider PVC: Platform Variant Creator PA: Platform Architect
PI: Platform Integrator

Table 1: Role Constraints for Service-based Platform Integration

Category 15 (i.e., "Decide integration of appropriate platforms") we list exemplary architectural decisions regarding the integration of heterogeneous domain platforms to the VSP:

- Decide on the type of component for integrating the platform service into the VSP.
- Decide on the connection of heterogeneous systems (in terms of synchronization and queuing behavior).
- Decide on the protocol(s) for accessing the VSP from the integrating platforms.
- Decide on how to accommodate diverse protocol requirements of integrating platforms.

For each Decision Category, the report [44] also provides governance rules, i.e., a detailed description of the rights and duties of each involved stakeholder role. As these stakeholders and stakeholder roles belong to different organizations and domains, collaboratively deciding and making ADDs while complying to all governance rules is very complicated and error-prone. Using our previously described approach for supporting decision making and documentation (see Section 4), we can precisely formalize the described governance rules in the form of decision making constraints (see Section 4.2) and rely on CoCoADvISE's automatic constraint enforcement capabilities in order to stay compliant to these governance rules. For instance, the governance rules for Decision Category 1 from Table 1 can be formalized and enforced by defining a Responsible Role constraint (i.e., C_{R_R}), with a (responsible) stakeholder role of "Platform Provider" (i.e., PP) and a corresponding Role Unanimity constraint (i.e., C_{U_R}). At runtime, these two constraints will enforce, that only a stakeholder with the role PP will be responsible for deciding on variability modeling (i.e., Responsible Role) and that the other participating stakeholders with the role PP have to decide unanimously (i.e., Role Unanimity) on this matter. In a similar way, our approach can be used to enforce the governance rules of the remaining Decision Categories 2–16 too (as can be seen in Table 1).

6. Empirical Evaluation

In order to collect empirical evidence about the effectiveness and efficiency of our proposed concepts, we conducted a controlled experiment with computer science students. We designed and executed our controlled experiment following the guidelines of Kitchenham [17] and analyzed and evaluated the results according to Wohlin et al.'s advice [61]. The following subsections discuss the goals and hypotheses of the controlled experiment, as well as its design and execution in detail.

6.1. Goals and Hypotheses

The goal of the experiment is twofold. On the one hand, we want to study and quantify the benefits of automatically enforcing constraints in a collaborative architectural decision making tool. On the other hand, we want to analyze and quantify the adverse effects of constraint violations in detail. Consequently, we postulate the following hypotheses:

Automatic enforcement of constraints in CoCoADvISE...

H_{01} has no effect or decreases the *effectiveness* of its users.

H_1 increases the *effectiveness* of its users.

H_{02} has no effect or decreases the *time related efficiency* of its users.

H_2 increases the *time related efficiency* of its users.

H_{03} has no effect or decreases the *effort related efficiency* of its users.

H_3 increases the *effort related efficiency* of its users.

We expect that the corresponding null hypotheses can be rejected. That is, we expect that automatic enforcement of constraints in CoCoADvISE increases both the *effectiveness* and the *time and effort related efficiency* of its users. In particular, we expect that users will manage to achieve more of the imposed work tasks than users that can not rely on automatic enforcement of constraints. We also expect that the former will have to invest both less effort (i.e., by performing less work steps/actions) and less time in order to achieve the same results.

6.2. Parameters and Values

During the experiment several dependent and independent variables have been observed. Table 2 provides a detailed description, including the type, scale type, unit and range of those variables.

Dependent Variables. All dependent variables have been extracted automatically from CoCoADvISE's database. In particular, we instrumented its source code in such a way that we could precisely record all user activities within the system. The variable *time* indicates a single user's total time spent logged in (i.e., the sum of all session durations). Similarly, the variable *actions* counts a user's total number of essential actions or work steps within the system. In particular, we consider the following actions in CoCoADvISE to be essential: create/remove a questionnaire, generate/remove/copy a decision, and select an option to a question. The variable *violations* indicates how many violations of decision making constraints a single user has caused. Finally, the variable *work* represents how much of the required work tasks in percent could actually be achieved. For instance, users that completed 3 out of 6 work tasks got a value of 50 for their *work* variable. The concrete number of work tasks depends on the role that is randomly assigned to each user. That is, the role *Software Architect* had to perform 7 tasks and the *Application Developer* 6, respectively.

Derived Variables. To allow for a meaningful comparison of *time* and *actions*, we decided to introduce two additional derived variables: $timeNorm$ and $actionsNorm$. In particular, we normalize the *time* and *actions* variables by dividing them by *work*. As a result, $timeNorm$ can be interpreted as the total time a user would have needed to finish all work tasks. Comparing, for instance, $timeNorm$ instead of *time*, rules out the possibility that the participants of one treatment group needed less time only because they "worked less" (i.e., in terms of *work*) than the participants of the other group.

Independent Variables. The independent variables *group*, *exp* and *commExp* can potentially influence the dependent variables. In particular, *group* contains a participant's treatment group, and *exp* and *commExp* concern their programming experience and commercial programming experience, respectively.

Type	Name	Description	Scale Type	Unit	Range
Dependent	*time*	Overall time needed to make and document decisions	Ratio	Minutes	Positive natural numbers including 0
	actions	Number of actions performed	Ratio	–	Positive natural numbers including 0
	violations	Number of constraint violations caused	Ratio	–	Positive natural numbers including 0
	work	Percentage of work that a single user is supposed to perform	Ratio	–	0 (lowest) to 100 (highest)
Derived	$timeNorm$ $(= \frac{time}{work})$	Time that would be needed to perform 100% of *work*	Ratio	Minutes	Positive natural numbers including 0
	$actionsNorm$ $(= \frac{actions}{work})$	Number of actions that would need to be performed to accomplish 100% of *work*	Ratio	–	Positive natural numbers including 0
Independent	*group*	Treatment group	Nominal	–	Either "experiment" or "control"
	exp	Programming experience	Ordinal	Years	4 classes: 0-1, 1-3, 3-6, >6
	commExp	Commercial programming experience in industry	Ordinal	Years	4 classes: 0-1, 1-3, 3-6, >6

Table 2: Observed and Derived Variables

6.3. Experiment Design

The controlled experiment was conducted in the context of Information System Technologies lecture at the Faculty of Computer Science, University of Vienna, Austria, in January 2014.

Participants. From the 48 students of the lecture, 26 participated in the control group and 22 in the experiment group, in teams of two students. The experiment was part of a practical exercise on architectural decisions for service-based software systems. The practical exercises took place in four separate groups (in different rooms) to which the students were randomly assigned. All students had background in Java programming, Web services, and design patterns.

Objects. As the basis for making and documenting pattern-based architectural decisions collaboratively and remotely, a list of documented architectural design patterns, as well as a set of reusable architectural decision models, were provided in the CoCoADvISE tool. The design patterns and architectural decision models were selected based on the lecture materials known to the students and the students' experiences from the previous practical exercises.

Instrumentation. In the preparation phase, all participants were given an introduction to CoCoADvISE and were asked to study the catalog of architectural design patterns and related technologies. Before starting with the experiment tasks, all participants had to fill in a short questionnaire regarding their programming experiences. Afterwards, all participants were provided with a description and requirements of the system to be designed ("An Online Retailer for Selling Books and Gadgets"), a description of the different stakeholder roles, their responsibilities, as well as a description of additional constraints regarding the collaborative decision making process. In Table 3 we give an example of the software system's requirements, and in Table 4 we summarize the two stakeholder roles along with an excerpt of their privileges and responsibilities. Note that in reality a larger number of roles would be needed, but in order to reach a controlled environment we had to simplify the roles used in the experiment. In total, the students had to consider three groups of requirements (*Expose Web Services to a Web Shop*, *Customer Login*, and *Manage Different Formats for Inventories*), given in descriptive form. Additionally, some hints were provided with information about the concrete decisions that were expected. Each requirement had to be covered by one or more documented architectural decisions.

Eventually, all participants were given access to the CoCoADvISE tool. For the needs of the controlled experiment a detailed list of related architectural design patterns and three reusable architectural decision models with use instructions were provided in the tool. The functionality of CoCoADvISE is described in Section 4 and the setting provided to the students can be viewed at `https://piri.swa.univie.ac.at/cocoadvise`[16]. The participants needed to

[16] Use the following user names (no password required): experiment1, experiment2, control1 or control2.

Name	Description
Customer Login	A customer needs to login in order to purchase books and gadgets online and is responsible for saving his/her session in order to keep the state of his/her orders (stateful remote objects). If the session is inactive for a predefined time period the session should expire and the customer gets automatically logged out. *Decide* how to implement the creation and lifecycle management of the sessions. **Hint:** Create a questionnaire based on the Resource and Lifecycle Management decision model.

Table 3: Online Retailer Requirement Example

Software Architect	The Software Architect is responsible for high-level decisions rather than for implementation details.
Application Developer	The Application Developer is responsible for decisions that refer to low-level design and implementation details.
Privileges	Only the Software Architect should be able to create the questionnaires giving a name already agreed/edited by both the Software Architect and the Application Developer. Also, only the creator of questionnaires and decisions is able to remove them, i.e., you are not allowed to delete questionnaires or decisions of your partner. [...] The Software Architect will make the final decision (generate decision) about the type of Web Service that will be exposed by the Online Retailer (as well as the transport protocol) but has to agree with the Application Developer on this before he/she makes the final decision. The same applies for the architectural decision regarding the service discovery. Once the decision about the type of web service has been made, the Application Developer can proceed with deciding the security and encryption of the web service. Only the Application Developer is responsible for deciding on security issues and the Software Architect should not interfere in this issue. [...]

Table 4: Decision Making Roles with their Privileges

reuse three architectural decision models (*Resource and Lifecycle Management*, *Message Transformations*, and *Web Services*) in teams, in order to make and document the architectural decisions related to the given requirements.

The crucial difference between the experiment and the control group was that we completely disabled the automatic constraint enforcement functionality for users belonging to the control group. In other words, the control group's members were completely responsible on their own for working in a way that no constraints are violated. In particular, they had to detect violations on their own and they also had to resolve them "manually". In contrast, these tasks have been automated for the experiment group.

Note that for the purpose of this experiment only the following constraint types have been considered: Unanimity, ResponsibleRole and RequiredPermission.

6.4. Execution

As described in the previous section, the experiment was executed in the context of the Information System Technologies lecture at the Faculty of Computer Science, University of Vienna in the Winter Semester 2013/2014.

The participants were randomly divided into groups of two persons and also randomly assigned to experiment and control group. As two of the four course groups with different numbers of students took place simultaneously at a time and the collaborating students were not allowed to work in the same room, we had to divide the participants in unequal groups of 22 (experiment group) and 26 (control group) participants. Three participants of the experiment group were excluded, as they did not hand in any results (that is, they did not edit any questionnaires or decisions).

As we can see in Figure 11 the programming experience, as well as the industry programming experience of the participants are comparable in both treatment groups with the control group having slightly longer programming experience and most of the students having more than 2.5 years of programming experience but very few having experience in industrial projects (0-1 years).

The same materials were handed out to all participants in the beginning of the exercise. The experiment group used the different version of CoCoAD-vISE where the constraints (such as the ones in Table 4) were integrated and automatically enforced (i.e., they could not be violated in any way).

The experiment was executed in two sessions of 90 minutes. In this time period, the students had to read, understand and execute the exercise. They were allowed to finish with the tasks earlier. Access to the tool was given only during these sessions to avoid offline work or discussion among students.

The collection of the participants' data has been performed automatically during the experiment. In particular, all relevant information, such as created questionnaires, selected options, and exchanged messages, as well as all relevant events, such as deletions or modifications of architectural decisions and changes of selected options were saved in a database.

No deviations from the initial study design occurred and no situations in which participants behaved unexpectedly.

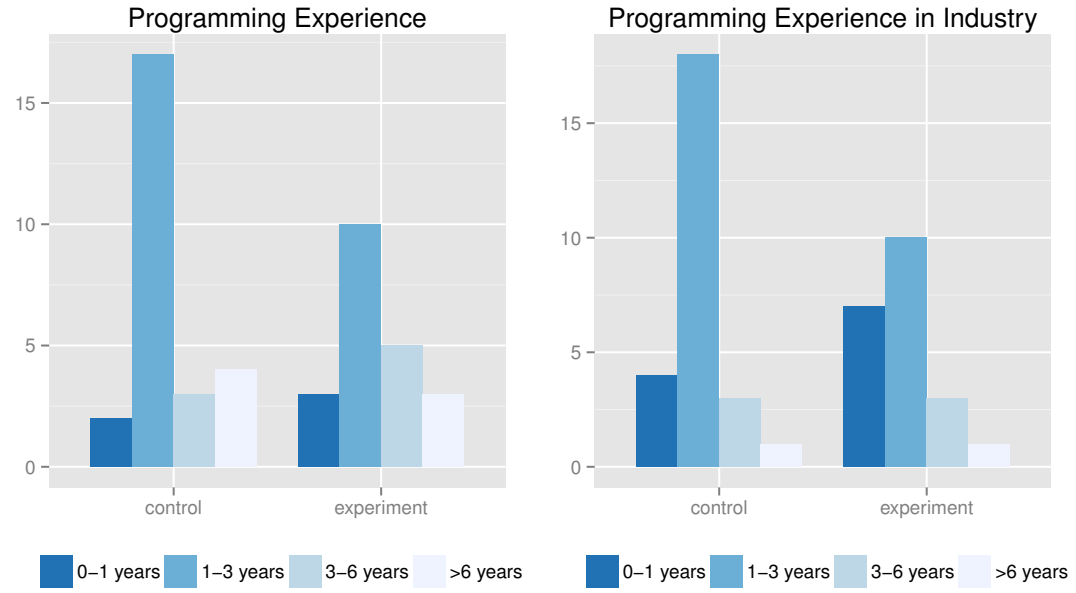


Figure 11: Participants' Programming Experience

7. Analysis of Results

The following statistical analysis has been carried out using R language and environment for statistical computing [51]. Note that the raw data as well as the corresponding R script for calculating these results are available online at `https://piri.swa.univie.ac.at/cocoadvise-experiment`.

7.1. Descriptive Statistics

As a first step in our analysis, we use descriptive statistics to compare observed variables related to the efficiency and the effectiveness of making and documenting architectural decisions. That is, Table 5 and Figure 12 display the mean and median values for the number of actions the participants of each treatment group needed to perform in order to complete the exercise (*actions*), the total time they needed (*time*), the percentage of tasks they completed (*work*), and the number of constraint violations they caused (*violations*).

It is clearly noticeable that the experiment group spent less time working on the exercise and had to perform less actions than the control group. A user of the control group caused 5.23 constraint violations on average, which are automatically prevented for the experiment group by our tool. The experiment group could finish more work tasks than the control group (i.e., 85.03% vs. 71.79% on average). Given these results, it makes sense to take a closer look at our derived variables (i.e., $timeNorm$ and $actionsNorm$). We can see that the gap between both treatment groups gets wider when we look at these derived variables. That is, the experiment group would need roughly 41% less time and nearly 44% less actions in order to completely finish all required work tasks.

Variable	Means		Medians	
	control	**experiment**	**control**	**experiment**
time (min)	163.72	120.53	148.33	120.21
timeNorm (min)	244.25	146.72	221.41	142.57
actions	95.15	67.05	88.50	57.00
actionsNorm	140.81	80.08	127.20	60.00
violations	5.23	–	3.50	–
work (%)	71.79	85.03	71.43	85.71

Table 5: Means and Medians of Observed Variables

Constraint Type	Means	Medians
Unanimity	0.27	0.00
ResponsibleRole	3.38	2.00
RequiredPermission	1.85	0.00

Table 6: Observed Constraint Violations per User in the Control Group

Finally, Table 6 provides the mean and median values of the observed constraint violations per type and per user in the control group. We notice that, on average, ResponsibleRole constraints were violated 3.38 times per user, followed by RequiredPermission constraints, which were violated 1.85 times per user.

7.2. Data Set Reduction

Studying the deviations from the means for each of the four variables that we observed we noticed a few outliers, i.e., points that are either much higher or much lower than the mean values. As these potential candidate data points for exclusion correspond to different participants (for instance, a student delivered more required work in less time) these single outlier points do not necessarily make the participant an outlier. Thus, we decided to exclude only participants who did not perform any action and delivered 0% of the required work tasks, and who therefore would make the study results vulnerable. This was done, however, before the data analysis (see explanation in Section 6.4); at this stage, we did not perform any further data set reduction.

7.3. Hypotheses Testing

Testing for Normal Distribution. In order to see whether we can apply parametric tests like the t-test that assume the normal distribution of the analyzed data, we tested the normality of the data by applying the Shapiro-Wilk test [43]. The null hypothesis of the Shapiro-Wilk test states that the input data is normally distributed. It is tested at the significance level of $\alpha = 0.05$ (i.e., the level of confidence is 95%). That is, if the calculated p-value is lower than 0.05 the null hypothesis is rejected and the input data is not normally distributed. If the

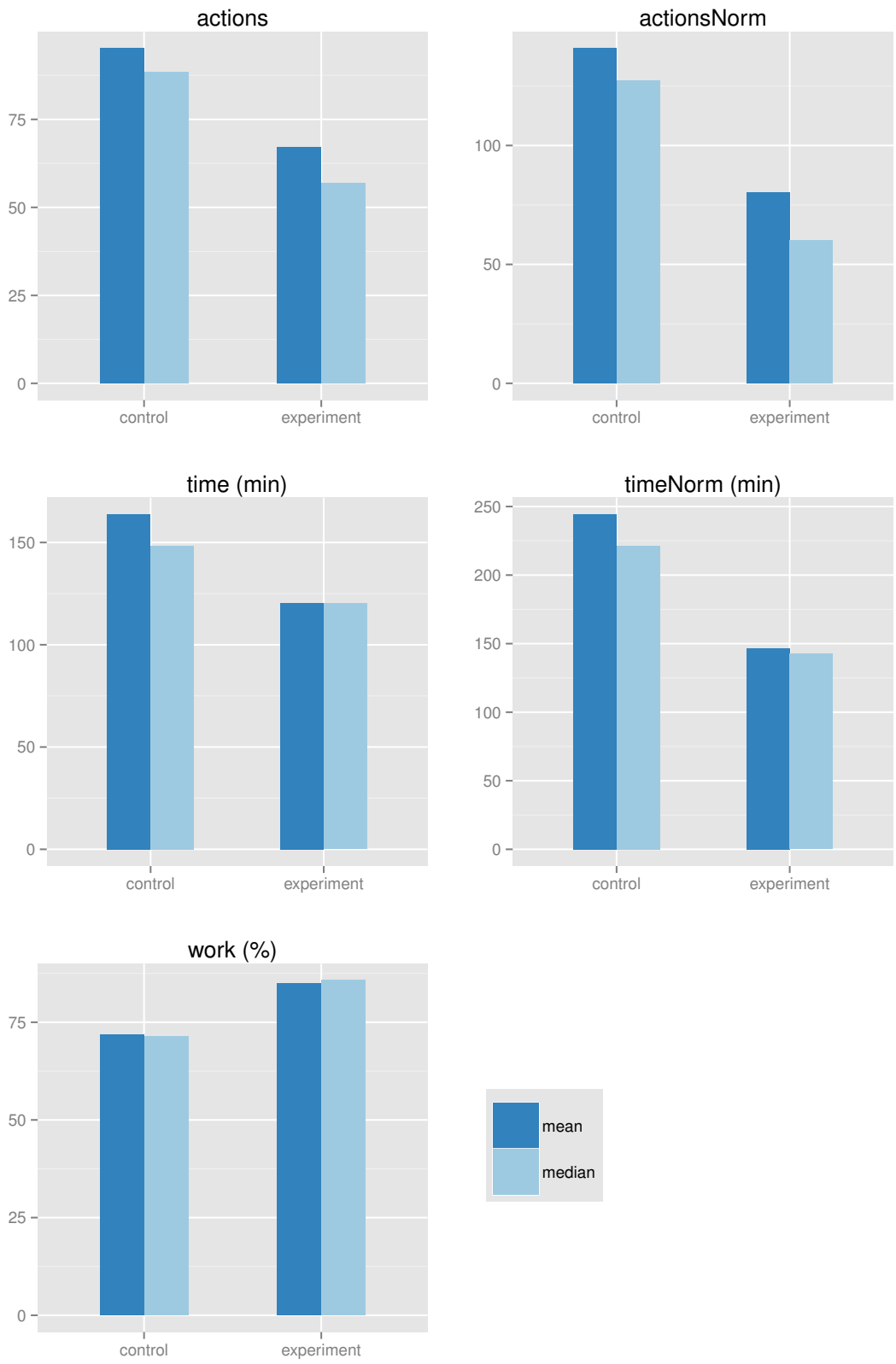


Figure 12: Means and Medians of Observed Variables

Variable	p-Value	
	control	**experiment**
time	0.0571	0.8451
actions	0.0092	0.0017
violations	0.0045	–
work	0.0850	0.0047

Table 7: Shapiro-Wilk Normality Test

p-value is higher than 0.05, we can not reject the null hypothesis that the data is normally distributed.

Table 7 lists the p-values of the Shapiro-Wilk normality test for each observed variable and treatment group. We can see that only *time* exhibits a very weak tendency of being normally distributed, while for all other variables it can not be concluded that they are normally distributed. As a result, we decided to pursue non-parametric statistical tests with our data.

Comparing the Means of Variables. To compare the means of variables, we applied the Wilcoxon rank-sum test [25]. The one-tailed Wilcoxon rank-sum test is a non-parametric test for assessing whether one of two data samples of independent observations is stochastically greater than the other. Its null hypothesis, which is appropriate for the hypotheses in our experiment, is that the means of the first variable's distribution is less than or equal to the means of the second variable's distribution, so that we can write $H_0 : A \leq B$. The Wilcoxon rank-sum test tries to find a location shift in the distributions, i.e., the difference in means of two distributions. The corresponding alternative hypothesis H_A could be written as $H_A : A > B$. If a p-value for the test is smaller than 0.05 (i.e., the level of confidence is 95%), the null hypothesis is rejected and the distributions are shifted. If a p-value is larger than 0.05, the null hypothesis can not be rejected, and we can not claim that there is a shift between the two distributions.

Table 8 contains the p-values of five Wilcoxon rank-sum tests that were performed to test our hypotheses (see Section 6.1). It also contains the corresponding null hypotheses (e.g., $\mathbf{H_{01}}$ is the null hypothesis of $\mathbf{H_1}$) and their assumptions regarding the means of a specific variable. Based on the obtained p-values, we can assess that all distributions show a statistically significant shift between each other and that all null hypotheses can be rejected.

Testing Hypothesis $\mathbf{H_1}$. In our experiment, we observed that the experiment group was able to perform more work tasks than the control group, i.e., their participants were *more effective* than the participants of the other group. With a p-value of 0.0093 we can *reject* the null hypothesis $\mathbf{H_{01}}$ (i.e., automatic enforcement of constraints in CoCoADvISE has no effect or decreases the *effectiveness* of its users). Hence, we can *accept* $\mathbf{H_1}$.

Hypothesis	Assumption	Variable (μ)	p-Value
$\mathbf{H_{01}}$	$\mu_{exp} \leq \mu_{control}$	$work$	0.0093
$\mathbf{H_{02}}$	$\mu_{exp} \geq \mu_{control}$	$time$	0.0040
		$timeNorm$	0.0003
$\mathbf{H_{03}}$	$\mu_{exp} \geq \mu_{control}$	$actions$	0.0018
		$actionsNorm$	0.0001

Table 8: Hypothesis Testing Results

That is, there is evidence that the automatic enforcement of constraints increases the *effectiveness* of its users.

Testing Hypothesis $\mathbf{H_2}$. We also found that the experiment group needed less time than the control group, i.e., its members were *more efficient* in terms of *time invested* than the members of the other group. This observation holds for both the observed variable *time* and the derived variable $timeNorm$. Hence, we tested the null hypothesis $\mathbf{H_{02}}$ (i.e., automatic enforcement of constraints in CoCoADvISE has no effect or decreases the *time related efficiency* of its users) for both variables. As both p-values were below 0.05 (i.e., 0.0040 in the case of *time* and 0.0003 in the case of $timeNorm$) we can *reject* $\mathbf{H_{02}}$ and *accept* $\mathbf{H_2}$.

That is, there is evidence that automatic enforcement of constraints increases the *time related efficiency* of its users.

Testing Hypothesis $\mathbf{H_3}$. Finally, we discovered that the experiment group performed less actions than the control group, i.e., its participants were *more efficient* in terms of *effort invested* than the participants of the other group. As this observation holds for both *actions* and $actionsNorm$ we tested the null hypothesis $\mathbf{H_{02}}$ (i.e., automatic enforcement of constraints in CoCoADvISE has no effect or decreases the *effort related efficiency* of its users) for both variables. The p-values of 0.0018 and 0.0001 led us to *reject* $\mathbf{H_{03}}$ and *accept* $\mathbf{H_3}$.

Hence, we conclude that there is evidence that automatic enforcement of constraints also increases the *effort related efficiency* of its users.

7.4. Regression Analysis

In order to better understand the adverse effects of constraint violations, this section presents a linear regression analysis. Table 9 depicts three different linear regression models that can be used to quantify the effect of constraint violations (*violations*) on a user's effectiveness ($work$) and efficiency (*actions* and *time*). In particular, it shows the explained variable, the value of the coefficient (i.e., *violations*) and the p-value of the corresponding (two-tailed) t-test, the regression's R^2 and the p-value of the corresponding F-test as well as the number of outliers that had to be excluded from the regression.

The null hypothesis of the t-test assumes that the corresponding coefficient has a value of 0. At the significance level of $\alpha = 0.05$, a p-value lower than 0.05 provides evidence that the coefficient is significantly different from 0. R^2, the

Variable	Coefficient (*violations*)		R^2	p-Value	# Outliers
	Value	p-Value			
time	5.6230	0.0039	0.1671	0.0039	0
actions	4.1728	0.0000006	0.4596	0.0000006	5
work	−1.2032	0.0468	0.0868	0.0468	2

Table 9: Linear Regression Models

coefficient of determination, is used as an indicator of how well a linear regression fits a set of data. The statistical test of significance for R^2 is the F-test. Linear regressions require the following four crucial assumptions to hold: Linearity, Homoscedasticity, Uncorrelatedness and Normality. Pena et al. proposed a procedure for testing these assumptions [35]. We used the corresponding `gvlma` R package for assuring (at a significance level of $\alpha = 0.05$) that the four assumptions hold for our linear regressions. Note that we had to iteratively increase the number of outliers to be excluded from the regressions until the `gvlma` package confirmed that all assumptions are justified. Eventually we had to exclude 5 participants from the regression that explains *actions* and 2 for the regression explaining *work*.

As we can see, the p-values of all t-tests and F-tests are below 0.05. Hence, we consider all coefficient values and the R^2 for each regression to be statistically significant at the significance level of $\alpha = 0.05$. Each regression model can be used to quantify the effect of a single constraint violation on each of the explained variables. For instance, regarding *time* our model predicts that a single constraint violation increases the overall time needed to make and document decisions by nearly 6 minutes. Similarly, a violation increases *actions*, i.e., the number of actions that a user performs, by roughly 4. With an R^2 of 0.4596 it can also be noted, that the coefficient *violations* "can explain" nearly 46% of *actions* variability. Finally, there is a negative relation between *violations* and *work*. According to our model, a single constraint violation reduces the percentage of work tasks that a user manages to accomplish by roughly 1.2%.

In summary, these regression models fortify and complement our main findings. While our hypotheses dealt with finding evidence that automatic enforcement of constraints is beneficiary in terms of effectivity and efficiency in general, the regression models provide further insights into (1) what exactly are the adverse effects of constraint violations and (2) to which extent they influence the effectivity and efficiency of users.

8. Discussion

The following subsections discuss our main findings and their implications as well as their threats to validity.

8.1. Evaluation of Results and Implications

Increased Effectiveness. Hypothesis $\mathbf{H_1}$ and the corresponding null hypothesis $\mathbf{H_{01}}$ concern the effectiveness of users of collaborative and constrained architectural decision making tools. In Section 7.3 we could provide evidence that the null hypothesis $\mathbf{H_{01}}$ can be rejected. Thus, automatic enforcement of constraints increases the effectiveness of its users.

We interpret this finding as follows. The concrete set of work tasks a specific user has to complete stems from the role that has been assigned to the user. In our experiment, the duties of each role have been described textually, as can be seen in Table 4. For instance, a concrete work task of users with the role *Software Architect* is that they are supposed to generate the decision about the type of Web service to be deployed. If the other user generates the decision instead, we do not increment the number of successfully accomplished work tasks of the *Software Architect*. In Table 6 we can see that ResponsibleRole constraints were violated 3.38 times per user (on average). Thus, it seems that many users were unsure or confused about who is supposed to do what in the decision making process. In fact, similar issues have been documented in [28] and [1].

As many users performed tasks that were supposed to be performed by another user and the fact that our approach for calculating the percentage of accomplished work penalizes these "deviations from the prescribed regulations", we conclude that our experiment provides evidence that automatic enforcement of constraints increases the effectiveness of its users.

Increased Efficiency. In Section 7.3 we could provide evidence that both null hypotheses $\mathbf{H_{02}}$ and $\mathbf{H_{03}}$ can be rejected. The corresponding alternative hypotheses $\mathbf{H_2}$ and $\mathbf{H_3}$ concern the efficiency of users of collaborative and constrained architectural decision making tools. Thus, automatic enforcement of constraints increases the efficiency of its users. To be exact, it increases both the time and effort related efficiency.

We have the following explanation for these findings. In order to be able to "manually" detect and prevent constraint violations, users have to read and understand the description and meaning of each defined constraint type first. Then, during working on their work tasks and performing actions, they have to be careful not to (unintentionally) cause constraint violations. In case a violation happens anyway, there are two possibilities. If the violation gets detected, the users have to resolve the violation, e.g., by revoking and redoing already performed tasks. In general, detecting and resolving constraints requires an additional investment of both time and effort. Violations which are not detected decrease – among other things – the effectiveness of users. To this end, especially our linear regression model (see Section 7.4) can be interpreted as a good estimator for predicting the effort reduction to be expected when introducing automatic enforcement of constraints. For instance, if we would expect an average of 5.23 constraint violations per user (which is the actual observed mean for *violations* in our experiment), our model predicts that we can anticipate our users to require 21.82 additional work steps needed to resolve these viola-

tions again. Analogously, these additional work steps are predicted to cost 34.64 additional minutes.

In summary, we can conclude that our experiment provides evidence that automatic enforcement of constraints increases the efficiency of users, because it takes away the burden of detecting, preventing and resolving constraint violations "manually" from the user.

Initial Development and Modeling Effort. A possible limitation of our approach is that automatic enforcement of decision making constraints, as proposed in this paper, is only possible if the required amount of time and effort gets invested into modeling decision and constraint specifications before the tool is used. In addition, in the rare case that a new constraint type is introduced, developers have to augment the tool with additional constraint checking and enforcement logic. As our approach is based on *reusable* architectural decision models that are supposed to be instantiated more than once, the modeling effort is only required once per reusable decision model and constraint type. Nowak et al. envision the idea of collaboration and knowledge exchange between different architectural knowledge repositories (i.e., repositories containing reusable architectural decision models) [32]. When applied to our context, this means a further reduction of initial modeling effort. Thus, models are shared, reused and adapted instead of built from scratch. Hence, except for rarely used decisions or constraint types this limitation should be negligible.

8.2. Threats to Validity

To ensure the validity of our results, we consider the categorization of validity threats of Wohlin [61] and discuss each of them separately in the context of our controlled experiment.

Conclusion Validity. The conclusion validity focuses on the relationship between the treatment we used in the experiment and the actual outcome, i.e., on the existence of a significant statistical relationship.

The way we measured the working time of the students automatically from the database entries may pose a threat to conclusion validity, as users might have spent some observed working time idle or with other tasks, or they might have worked offline without the system noticing the working time. In addition, to measure the actual time spent on working with the CoCoADvISE tool is very difficult, if not impossible, as the participants may have spent some time reading the tasks or familiarizing with the tool. However, we think that idle working times, times spent on other tasks, or offline work can largely be excluded due to the limited experiment time of 180 minutes in which the participants needed to work in a concentrated manner in order to get the work done.

The number of participants (48 students) may also affect the statistical validity of the results.

Internal Validity. The internal validity refers to the extent to which treatment or independent variables caused the effects seen on the dependent variables.

In order to reduce this kind of validity threats, we made sure that the participants of both groups had at least medium experience in programming and design – with slight differences – and that they were aware of the architectural design patterns they had to use for making and documenting architectural decisions (they had also implemented some of them during the practical course before the experiment).

Also, the experiment was carried out in a controlled environment and the group members were in different rooms and did not know the identity of their partner. An observer in the room ensured that no interactions between the participants of the same room occurred. The students did not know the goals of the experiment or the group they belong to, nor could they realize that the control and experiment groups were working with different versions of CoCoADvISE.

Our inability, to effectively prevent the participants from using external Web sites (i.e., search engines, social networks, chats, wikis, etc.) during the experiment, might also pose a potential – but arguably negligible – threat to validity. We believe, that it is very unlikely, that any of these external Web sites might have been advantageous in actually preventing constraint violations.

A threat to validity was introduced by the execution of the experiment in two different sessions. To limit this threat, we did not allow any access to CoCoADvISE and the accompanying materials outside these two sessions.

Construct Validity. The construct validity focuses on the suitability of the experiment design for the theory behind the experiment and the observations.

The students worked in their task assignment only on a single software system with an excerpt of the full list of requirements. However, it is likely that this did not (heavily) affect the validity of the results because the constraints in the collaboration were our focus. We regard the amount and type of the constraints introduced in the experiment to be grounded in real-life collaborative architectural decision making scenarios.

The variables that have been observed in the experiment are regarded as accurate and objective as they are related to the actual use of tools and were automatically extracted from the database.

Also, for calculating the completed work per participant, we first extracted a list of required tasks from the exercise description which was afterwards used to calculate the completion of work automatically from the database entries.

External Validity. The external validity is concerned with whether the results are generalizable outside the scope of our study.

The subjects of the experiment have medium programming experience and were familiar with the architectural design decisions they were asked to make. However, only few students have experience in the industry. We hence consider the group under study to be representative for novice software developers or architects and plan to test the same hypotheses with other target groups as well.

Kitchenham et al. regard students close to practitioners, as they are considered to be the next generation of software professionals [17].

As mentioned before, the measurements were extracted from the tool database, avoiding any bias by the experimenters.

The system under study and the corresponding architectural decision models and patterns are representative for Web and enterprise information systems, and hence it is likely that the findings can be generalized to similar system domains and decision models. It would require additional experiments to determine if they can be generalized to vastly different system domains, such as software systems operating close to the hardware.

Finally, in our experiment we observed group decision making with groups of only two members. In our point of view, it is highly likely that the results are similar for slightly larger groups (e.g., of 3 or 4 members). However, it is unclear, if the results can be generalized to larger groups of decision makers.

8.3. Inferences

In principle, our observations are coherent with the findings of similar studies in slightly different contexts. For instance, Herbsleb et al. present a theory that models collaborative software engineering as a distributed constraint satisfaction problem [12]. They also found that backtracking, as a result of constraint violations, increases both the time and effort to be invested. This is a further indication that our findings should be generalizable. In particular, we believe that virtually any kind of collaborative process that concerns different stakeholder roles and demands to be restricted by various domain and context specific constraints will benefit from automatic constraint enforcement in a similar way to CoCoADvISE.

9. Conclusions and Future Work

The approach presented in this paper is the first one to consider the precise definition and automatic enforcement of constraints in real-time collaborative architectural decision making. CoCoADvISE ensures that stakeholders with different roles make and document collaborative architectural design decisions consistently. We demonstrate the applicability of our approach in an industrial context and with the help of a controlled experiment we are also able to report strong evidence that the automatic enforcement of constraints leads to increased time and effort related efficiency and effectiveness of the users while making and documenting architectural decisions.

We consider our approach and accompanying tool to be relevant and useful for other collaborative software engineering tools as well, which involve various stakeholder roles and distributed teams. Therefore, we plan to extend CoCoADvISE to cover other constrainable collaborative activities with focus on software architecture processes.

In our future work, we will also collect more empirical evidence about the supportive effect of automatic enforcement of constraints in collaborative architectural decision making tools on the efficiency and effectiveness of users, by

conducting similar controlled experiments. Our main goal is to test our assumptions with practitioners, receive feedback regarding the usability of our tool, and test our approach with different group sizes, in different system domains, and with different decision models.

Acknowledgments

We would like to thank all students of the Information System Technologies lecture in the Winter Semester 2013/2014 for participating in the experiment.

Appendix A. Generic Meta-model for Decision Making Constraints

This appendix provides the complete formal definition of the Constrainable Decision Meta-model introduced in Section 4.3. In particular, Definition A.1 provides a list of elements and their relations, Definition A.2 presents crucial model invariants to be considered at design time, and Definition A.3 lists model invariants relevant at execution time.

To provide a self-contained view in this paper, the following formal meta-model repeats the core definitions regarding the concepts of subjects, roles and permissions from [49], which form the basis for our approach.

Definition A.1 (Constrainable Decision Meta-model)
A Constrainable Decision Model $CDM = (E, M)$ where $E = DM \cup D \cup Q \cup O \cup S \cup R \cup P \cup DD \cup O_S \cup C$ refers to pairwise disjoint sets of the meta-model and $M = dma \cup qda \cup oqa \cup rsa \cup pra \cup dda \cup gs \cup gr \cup soa \cup ss \cup sr \cup cda \cup cma \cup rruca \cup ssuca \cup rrrca \cup srsca \cup prpca \cup cd$ to mappings that establish relationships, such that:

- For the sets of the meta-model:
  - An element of DM is called *Decision Model*. $DM \neq \emptyset$.
  - An element of D is called *Decision*. $D \neq \emptyset$.
  - An element of Q is called *Question*. $Q \neq \emptyset$.
  - An element of O is called *Option*. $O \neq \emptyset$.
  - An element of S is called *Subject*. $S \neq \emptyset$.
  - An element of R is called *Role*. $R \neq \emptyset$.
  - An element of P is called *Permission*. $P \supseteq \{selectOption, generateDecision\}$.
  - An element of DD is called *Documented Decision*. $DD \neq \emptyset$.
  - An element of O_S is called *Option Selection*.
  - An element of C is called *Constraint*. $C = C_U \cup C_{U_R} \cup C_{U_S} \cup C_{R_R} \cup C_{R_S} \cup C_P$
  - An element of C_U is called *Unanimity Constraint*.

- An element of C_{U_R} is called *Role Unanimity Constraint.*
- An element of C_{U_S} is called *Subject Unanimity Constraint.*
- An element of C_{R_R} is called *Responsible Role Constraint.*
- An element of C_{R_S} is called *Responsible Subject Constraint.*
- An element of C_P is called *Required Permission Constraint.*

In the list below, we iteratively define the partial mappings of the Decision Making Constraint Model and provide corresponding formalizations ($\mathcal{P}$ refers to the power set):

1. A decision model consists of many decisions and each decision belongs to exactly one decision model.
   Formally: The *injective* mapping $dma : DM \mapsto \mathcal{P}(D)$ is called **decision-to-decision-model assignment**. For $dma(dm) = D_{dm}$ we call $dm \in DM$ *decision model* and $D_{dm} \subseteq D$ is called the set of *decisions assigned to dm*. The mapping $dma^{-1} : D \mapsto DM$ returns the decision model a decision is assigned to.
2. A decision consists of many questions and each question belongs to exactly one decision.
   Formally: The *injective* mapping $qda : D \mapsto \mathcal{P}(Q)$ is called **questions-to-decision assignment**. For $qda(d) = Q_d$ we call $d \in D$ *decision* and $Q_d \subseteq Q$ is called the set of *questions assigned to d*. The mapping $qda^{-1} : Q \mapsto D$ returns the decision a question is assigned to.
3. A question provides many options and each option belongs to exactly one question.
   Formally: The *injective* mapping $oqa : Q \mapsto \mathcal{P}(O)$ is called **option-to-question assignment**. For $oqa(q) = O_q$ we call $q \in Q$ *question* and $O_q \subseteq O$ is called the set of *options assigned to q*. The mapping $oqa^{-1} : O \mapsto Q$ returns the question an option is assigned to.
4. Roles are assigned to subjects (i.e., human users), and through their roles the subjects acquire the rights to perform certain tasks (see [49]). The role-to-subject assignment relation is a many-to-many relation, so that each subject may own several roles and each role can be assigned to different subjects. For example, in case the "Software Architect" role is assigned to two subjects called Alice and Bob, both can perform all tasks assigned to the "Software Architect" role.
   Formally: The *injective* mapping $rsa : S \mapsto \mathcal{P}(R)$ is called **role-to-subject assignment**. For $rsa(s) = R_s$ we call $s \in S$ *subject* and $R_s \subseteq R$ the set of *roles assigned to this subject* (the set of roles owned by s). The mapping $rsa^{-1} : R \mapsto \mathcal{P}(S)$ returns all subjects assigned to a role (the set of subjects owning a role).
5. Permissions are assigned to roles. The permission-to-role assignment relation is a many-to-many relation, so that each role may own several permissions and each permission can be assigned to different roles.

Formally: The *injective* mapping $pra : R \mapsto \mathcal{P}(P)$ is called **permission-to-role assignment**. For $pra(r) = P_r$ we call $r \in R$ *role* and $P_r \subseteq P$ the set of *permissions assigned to this role* (the set of permissions owned by r). The mapping $pra^{-1} : P \mapsto \mathcal{P}(R)$ returns all roles assigned to a permission (the set of roles owning a permission).

6. At runtime, users can generate documented decisions which are based on a reusable decision. Thus, when a user generates a documented decision, a new documented decision is assigned to the corresponding (reusable) decision (see Figure 2).
   Formally: The mapping $dda : D \mapsto \mathcal{P}(DD)$ is called **documented-decision-to-decision assignment**. For $dda(d) = DD_d$ we call $d \in D$ *decision* and $DD_d \subseteq DD$ is called the set of *documented decisions assigned to d*.
7. As defined in Definition A.1.6, documented decisions are assigned to decisions whenever users generate documented decisions. The generating-subject mapping is used to hold the exact subject that generated a documented decision.
   Formally: The mapping $gs : DD \mapsto S$ is called **generating-subject mapping**. For $gs(dd) = s$ we call $s \in S$ the *generating subject* and $dd \subseteq DD_S$ is called the *documented decision*.
8. Similarly to Definition A.1.7, we define the role that is used to generate a documented decision to be called the generating-role of the corresponding documented decision.
   Formally: The mapping $gr : DD \mapsto R$ is called **generating-role mapping**. For $gr(dd) = r$ we call $r \in R$ the *generating role* and $dd \subseteq DD_S$ is called the *documented decision*.
9. At runtime, users of a decision model can select options. Thus, when a user selects an option, a new option selection is assigned to the corresponding option (see Figure 6(b)).
   Formally: The mapping $soa : O \mapsto \mathcal{P}(O_S)$ is called **selection-to-option assignment**. For $soa(o) = O_{S_o}$ we call $o \in O$ *option* and $O_{S_o} \subseteq O_S$ is called the set of *option selections assigned to o*.
10. As defined in Definition A.1.9, option selections are assigned to options whenever users select options. The purpose of an option selection is to hold the subject and role that is used to select a certain option. For example, if subject Alice selects the option "only JSON", the corresponding option selection holds a reference to the subject "Alice".
    Formally: The mapping $ss : O_S \mapsto S$ is called **selecting-subject mapping**. For $ss(os) = s$ we call $s \in S$ the *selecting subject* and $os \subseteq O_S$ is called the *option selection*.
11. Similarly to Definition A.1.10, we define the role that is used to select a certain option to be called the selecting-role of the corresponding option selection.
    Formally: The mapping $sr : O_S \mapsto R$ is called **selecting-role mapping**. For $sr(os) = r$ we call $r \in R$ the *selecting role* and $os \subseteq O_S$ is called the *option selection*.

12. Particular reusable architectural decisions can be made subject to decision making constraints. For example, the decision "RESTful HTTP vs. SOAP/WS-*" (see Figure 3), which is required to be decided unanimously by all stakeholders, may be made subject to a unanimity constraint. More precisely, decisions are made subject to constraints by assigning them to constraints.
    Formally: The *injective* mapping $cda : D \mapsto \mathcal{P}(C)$ is called **constraint-to-decision assignment**. For $cda(d) = C_d$ we call $d \in D$ *decision* and $C_d \subseteq C$ is called the set of *constraints assigned to d*. The mapping $cda^{-1} : C \mapsto \mathcal{P}(D)$ returns the set of decisions a constraint is assigned to.
13. All reusable architectural decisions of a decision model can be made subject to decision making constraints at once. For instance, all decisions of a decision model can statically be made subject to a Required Permission constraint by assigning the corresponding constraint to the decision model.
    Formally: The *injective* mapping $cma : DM \mapsto \mathcal{P}(C)$ is called **constraint-to-decision-model assignment**. For $cma(dm) = C_{dm}$ we call $dm \in DM$ *decision model* and $C_{dm} \subseteq C$ is called the set of *constraints assigned to dm*. The mapping $cma^{-1} : C \mapsto \mathcal{P}(DM)$ returns the set of decision models a constraint is assigned to.
14. A Role Unanimity Constraint enforces that all subjects that own at least one of a specific set of roles have to unanimously agree on the same set of options, at runtime. Thus, these roles are assigned to role unanimity constraints. The role-to-role-unanimity-constraint assignment relation is a many-to-many relation, so that each role may be assigned to several role unanimity constraints and each role unanimity constraint can be assigned to different roles.
    Formally: The *injective* mapping $rruca : C_{U_R} \mapsto \mathcal{P}(R)$ is called **role-to-role-unanimity-constraint assignment**. For $rruca(ruc) = R_{ruc}$ we call $ruc \in C_{U_R}$ *role unanimity constraint* and $R_{ruc} \subseteq R$ the set of *roles assigned to this role unanimity constraint*. The mapping $rruca^{-1} : R \mapsto \mathcal{P}(C_{U_R})$ returns all role unanimity constraints assigned to a role.
15. A Subject Unanimity Constraint enforces that a specific set of subjects have to unanimously agree on the same set of options, at runtime. Thus, these subjects are assigned to subject unanimity constraints. The subject-to-subject-unanimity-constraint assignment relation is a many-to-many relation, so that each subject may be assigned to several subject unanimity constraints and each subject unanimity constraint can be assigned to different subjects.
    Formally: The *injective* mapping $ssuca : C_{U_S} \mapsto \mathcal{P}(S)$ is called **subject-to-subject-unanimity-constraint assignment**. For $ssuca(suc) = S_{suc}$ we call $suc \in C_{U_S}$ *subject unanimity constraint* and $S_{suc} \subseteq S$ the set of *subjects assigned to this subject unanimity constraint*. The mapping $ssuca^{-1} : S \mapsto \mathcal{P}(C_{U_S})$ returns all subject unanimity constraints assigned to a subject.
16. A Responsible Role Constraint enforces that only subjects that own at

least one of a specific set of roles shall be allowed to make and generate a specific decision. Thus, these roles are assigned to responsible role constraints. The role-to-responsible-role-constraint assignment relation is a many-to-many relation, so that each role may be assigned to several responsible role constraints and each responsible role constraint can be assigned to different roles.
Formally: The *injective* mapping $rrrca : C_{R_R} \mapsto \mathcal{P}(R)$ is called **role-to-responsible-role-constraint assignment**. For $rrrca(rrc) = R_{rrc}$ we call $rrc \in C_{R_R}$ *responsible role constraint* and $R_{rrc} \subseteq R$ the set of *roles assigned to this responsible role constraint*. The mapping $rrrca^{-1} : R \mapsto \mathcal{P}(C_{R_R})$ returns all responsible role constraints assigned to a role.

17. A Responsible Subject Constraint enforces that only a specific set of subjects shall be allowed to make and generate a specific decision. Thus, these subjects are assigned to responsible subject constraints. The subject-to-responsible-subject-constraint assignment relation is a many-to-many relation, so that each subject may be assigned to several responsible subject constraints and each responsible subject constraint can be assigned to different subjects.
Formally: The *injective* mapping $srsca : C_{R_S} \mapsto \mathcal{P}(S)$ is called **subject-to-responsible-subject-constraint assignment**. For $srsca(rsc) = S_{rsc}$ we call $rsc \in C_{R_S}$ *responsible subject constraint* and $S_{rsc} \subseteq S$ the set of *subjects assigned to this responsible subject constraint*. The mapping $srsca^{-1} : S \mapsto \mathcal{P}(C_{R_S})$ returns all responsible subject constraints assigned to a subject.
18. A Required Permission Constraint enforces that only subjects that own certain permissions (i.e., via their roles) within the system are allowed to perform the corresponding activities. For example, consider permission "Generate Decision" is only assigned to role "Software Architect", then only subjects that own the role "Software Architect" shall be allowed to generate decisions. The permissions to be enforced have to be assigned to required permission constraints. The permission-to-required-permission-constraint assignment relation is a many-to-many relation, so that each permission may be assigned to several required permission constraints and each required permission constraint can be assigned to different permissions.
Formally: The *injective* mapping $prpca : C_P \mapsto \mathcal{P}(P)$ is called **permission-to-required-permission-constraint assignment**.
For $prpca(rpc) = P_{rpc}$ we call $rpc \in C_P$ *required permission constraint* and $P_{rpc} \subseteq P$ the set of *permissions assigned to this required permission constraint*. The mapping $prpca^{-1} : P \mapsto \mathcal{P}(C_P)$ returns all required permission constraints assigned to a permission.
19. A decision can effectively be constrained either by a constraint that is directly assigned to the decision (i.e., using a constraint-to-decision assignment, see, Definition A.1.12), or by a constraint that is assigned to the corresponding decision model (i.e., using a constraint-to-decision-model assignment, see, Definition A.1.13). The constrained-decision mapping

aggregates the set of decisions that are constrained by a constraint. Formally: The mapping $cd : C \mapsto \mathcal{P}(D)$ is called **constrained-decision mapping**, such that for each constraint $c \in C$ the set of decisions that are effectively constrained by c are returned, i.e., $cd(c) = dma(cma^{-1}(c)) \cup cda^{-1}(c)$.

Definition A.2 (Design Time *CDM* Meta-model Invariants)
Let $CDM = (E, M)$ be a Constrainable Decision Model. CDM is said to be statically correct if the following requirements hold:

1. A constraint either constrains a single decision or a complete decision model. It is either assigned to a decision or a decision model. Therefore:

$$\forall c \in C(cda^{-1}(c) = \emptyset \oplus cma^{-1}(c) = \emptyset)$$

   Note that the $\oplus$ symbol represents the XOR (i.e., exclusive or) operation.

2. All roles that are assigned to a responsible role constraint must own the permission "generate decision". Therefore:

$$\forall r \in R(rrrca^{-1}(r) \neq \emptyset \Rightarrow generateDecision \in pra(r))$$

3. All subjects that are assigned to a responsible subject constraint must own a role that owns the permission "generate decision". Therefore:

$$\forall s \in S(srsca^{-1}(s) \neq \emptyset \Rightarrow \exists r \in rsa(s)(generateDecision \in pra(r)))$$

4. Each permission that is assigned to a required permission constraint must be owned by at least one role. Therefore:

$$\forall c \in C_P, p \in prpca(c)(pra^{-1}(p) \neq \emptyset)$$

5. When an unanimity constraint is used there must be at least one subject that has the permission "select option". Therefore:

$$\forall c \in C_U \Rightarrow \exists r \in R(selectOption \in pra(r))$$

6. All roles that are assigned to role unanimity constraints must have the permission "select option". Therefore:

$$\forall r \in R(rruca^{-1}(r) \neq \emptyset \Rightarrow selectOption \in pra(r))$$

7. All subjects that are assigned to subject unanimity constraints must have the permission "select option". Therefore:

$$\forall s \in S(ssuca^{-1}(s) \neq \emptyset \Rightarrow \exists r \in rsa(s)(selectOption \in pra(r)))$$

8. Subject unanimity constraints and role unanimity constraints may potentially conflict. More precisely, if both constraint types are used simultaneously, it is required that each subject assigned to the subject unanimity constraint owns each role assigned to the role unanimity constraint. Therefore:

$$\forall dm \in DM(\forall d \in dma(dm)(\forall c_r \in cda(d) \cup cma(dm)(\\ \forall c_s \in cda(d) \cup cma(dm)(c_r \in C_{U_R} \wedge c_s \in C_{U_S} \\ \Rightarrow \forall s \in ssuca(c_s)(rsa(s) \supseteq rruca(c_r))))))$$

Definition A.3 (Execution Time *CDM* Meta-model Invariants) Let $CDM = (E, M)$ be a Constrainable Decision Model. CDM is said to be dynamically correct if the following requirements hold:

1. For each option selection, the selecting subject must own the corresponding selecting role. Therefore:

$$\forall os \in O_S(sr(os) \in rsa(ss(os)))$$

2. A subject may only select a single option for each question. Therefore:

$$\forall q \in Q(\forall o \in oqa(q)(\forall os_x \in soa(o)(\forall os_y \in soa(o)(\\ os_x \neq os_y \wedge \exists ss(os_x) \Rightarrow ss(os_x) \neq ss(os_y)))))$$

3. All subjects must choose an option for each question. Therefore:

$$\forall q \in Q(\forall s \in S(\exists o \in oqa(q)(\exists os \in soa(o)(ss(os) = s))))$$

4. All decision making constraint type definitions (i.e., CT_1 through CT_6, see Section 4.3) must be met. Therefore:

$$\forall dm \in DM(CT_1 \wedge CT_2 \wedge CT_3 \wedge CT_4 \wedge CT_5 \wedge CT_6)$$

References

[1] N. D. Anh and D. S. Cruzes. Coordination of software development teams across organizational boundary – an exploratory study. In *8th IEEE International Conference on Global Software Engineering (ICGSE)*, pages 216–225, Aug 2013.

[2] M. A. Babar and I. Gorton. A Tool for Managing Software Architecture Knowledge. In *Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent*, SHARK-ADI'07, Washington, DC, USA, 2007. IEEE Computer Society.

[3] M. Cantor and J. D. Sanders. Operational IT governance. Technical report, IBM developerWorks, 2007.

[4] V. Clerc, E. de Vries, and P. Lago. Using wikis to support architectural knowledge management in global software development. In *Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge*, pages 37–43. ACM, 2010.

[5] Y. Dubinsky, A. Yaeli, and A. Kofman. Effective Management of Roles and Responsibilities: Driving Accountability in Software Development Teams. *IBM J. Res. Dev.*, 54(2):173–183, Mar. 2010.

[6] J. Eckstein. *Agile Software Development with Distributed Teams: Staying Agile in a Global World*. Dorset House, 2010.

[7] R. Farenhorst, P. Lago, and H. Van Vliet. Effective Tool Support for Architectural Knowledge Sharing. In *Proceedings of the First European Conference on Software Architecture*, ECSA'07, pages 123–138, Berlin, Heidelberg, 2007. Springer-Verlag.

[8] P. Gaubatz and U. Zdun. Supporting Entailment Constraints in the Context of Collaborative Web Applications. In *28th Symposium On Applied Computing*, pages 736–741, USA, March 2013. ACM.

[9] M. Goldman, G. Little, and R. C. Miller. Real-time collaborative coding in a web ide. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST'11, pages 155–164, New York, NY, USA, 2011. ACM.

[10] N. B. Harrison, P. Avgeriou, and U. Zdun. Using Patterns to Capture Architectural Decisions. *IEEE Software*, 24(4):38–45, 2007.

[11] J. D. Herbsleb and R. E. Grinter. Architectures, Coordination, and Distance: Conway's Law and Beyond. *IEEE Software*, 16(5):63–70, Sept. 1999.

[12] J. D. Herbsleb, A. Mockus, and J. A. Roberts. Collaboration in software engineering projects: A theory of coordination. In *Proceedings of the International Conference on Information Systems (ICIS)*, page 38, 2006.

[13] A. Jansen and J. Bosch. Software Architecture as a Set of Architectural Design Decisions. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 109–120. IEEE Computer Society, 2005.

[14] C. Jensen and W. Scacchi. Governance in Open Source Software Development Projects: A Comparative Multi-level Analysis. In *Open Source Software: New Horizons*, volume 319 of *IFIP Advances in Information and Communication Technology*, pages 130–142. Springer Berlin Heidelberg, 2010.

[15] M. Jensen and S. Feja. A security modeling approach for web-service-based business processes. In *16th Annual IEEE International Conference on the Engineering of Computer Based Systems (ECBS'09)*, pages 340–347, 2009.

[16] M. Kalumbilo. Effective Specification of Decision Rights and Accountabilities for Better Performing Software Engineering Projects. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE'12, pages 1503–1506, Piscataway, NJ, USA, 2012. IEEE Press.

[17] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg. Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734, Aug. 2002.

[18] A. Kofman, A. Yaeli, T. Klinger, and P. Tarr. Roles, Rights, and Responsibilities: Better Governance Through Decision Rights Automation. In *Proceedings of the 2009 ICSE Workshop on Software Development Governance*, SDG'09, pages 9–14, Washington, DC, USA, 2009. IEEE Computer Society.

[19] P. Kruchten. An Ontology of Architectural Design Decisions. In *Proceedings of 2nd Workshop on Software Variability Management*, pages 54–61, 2004.

[20] P. Liang, A. Jansen, and P. Avgeriou. Knowledge Architect: A Tool Suite for Managing Software Architecture Knowledge. Technical report, University of Groningen, 2009.

[21] I. Lytra, S. Sobernig, and U. Zdun. Architectural Decision Making for Service-Based Platform Integration: A Qualitative Multi-Method Study. In *Joint 10th Working IEEE/IFIP Conference on Software Architecture & 6th European Conference on Software Architecture (WICSA/ECSA), Helsinki, Finland*, pages 111–120. IEEE Computer Society, 2012.

[22] I. Lytra, H. Tran, and U. Zdun. Supporting Consistency Between Architectural Design Decisions and Component Models Through Reusable Architectural Knowledge Transformations. In *Proceedings of the 7th European Conference on Software Architecture (ECSA)*, ECSA'13, pages 224–239, Berlin, Heidelberg, 2013. Springer-Verlag.

[23] A. MacLean, R. Young, V. Bellotti, and T. Moran. Questions, Options, and Criteria: Elements of Design Space Analysis. *Human-Computer Interaction*, 6:201–250, 1991.

[24] P. Maheshwari and A. Teoh. Supporting ATAM with a collaborative Web-based software architecture evaluation tool. *Science of Computer Programming*, 57(1):109–128, 2005.

[25] H. B. Mann and W. D. R. On a Test of Whether One of Two Random Variables is Stochastically Larger than the Other. *Annals of Mathematical Statistics*, 18(1):50–60, 1947.

[26] C. Mayr, U. Zdun, and S. Dustdar. Reusable Architectural Decision Model for Model and Metadata Repositories. In F. de Boer, M. Bonsangue, and

E. Madelaine, editors, *Formal Methods for Components and Objects*, volume 5751 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin Heidelberg, 2009.

[27] C. Miesbauer and R. Weinreich. Classification of Design Decisions: An Expert Survey in Practice. In *Proceedings of the 7th European Conference on Software Architecture*, ECSA'13, pages 130–145, Berlin, Heidelberg, 2013. Springer-Verlag.

[28] A. Nakakawa, P. v. Bommel, and H. A. Proper. Challenges of involving stakeholders when creating enterprise architecture. In B. v. Dongen and H. Reijers, editors, *Proceedings of the 5th SIKS/BENAIS Conference on Enterprise Information Systems (EIS-2010), Eindhoven, The Netherlands*, pages 43–55, November 2010.

[29] A. Nakakawa, P. van Bommel, and H. A. Proper. Supplementing Enterprise Architecture Approaches with Support for Executing Collaborative Tasks - a Case of TOGAF ADM. *International Journal of Cooperative Information Systems*, 22(2), 2013.

[30] R. Nord, P. C. Clements, D. Emery, and R. Hilliard. A Structured Approach for Reviewing Architecture Documentation (CMU/SEI-2009-TN-030). Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2009.

[31] M. Nowak and C. Pautasso. Team Situational Awareness and Architectural Decision Making with the Software Architecture Warehouse. In *7th European Conference on Software Architecture*, ECSA'13, pages 146–161, Berlin, Heidelberg, 2013. Springer-Verlag.

[32] M. Nowak, C. Pautasso, and O. Zimmermann. Architectural Decision Modeling with Reuse: Challenges and Opportunities. In *Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge*, SHARK'10, pages 13–20, New York, NY, USA, 2010. ACM.

[33] Object Management Group. OMG Unified Modeling Language (OMG UML): Superstructure (Version 2.2). URL: `http://www.omg.org/spec/UML/2.2/`, February 2009.

[34] P. Ovaska, M. Rossi, and P. Marttiin. Architecture as a Coordination Tool in Multi-site Software Development. *Software Process: Improvement and Practice*, 8(4):233–247, Oct./Dec. 2003. Special Issue: Global Software Development: Growing Opportunities, Ongoing Challenges.

[35] E. A. Peña and E. H. Slate. Global Validation of Linear Model Assumptions. *Journal of the American Statistical Association*, 101(473):341–354, 2006.

[36] B. Raadt, S. Schouten, and H. Vliet. Stakeholder Perception of Enterprise Architecture. In *Proceedings of the 2nd European Conference on Software Architecture*, ECSA'08, pages 19–34, Berlin, Heidelberg, 2008. Springer.

[37] V. S. Rekha and H. Muccini. A Study on Group Decision-Making in Software Architecture. In *IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 185–194, 2014.

[38] S. Rekha V. and H. Muccini. Suitability of Software Architecture Decision Making Methods for Group Decisions. In *Software Architecture*, volume 8627 of *Lecture Notes in Computer Science*, pages 17–32. Springer International Publishing, 2014.

[39] N. Schuster, O. Zimmermann, and C. Pautasso. ADkwik: Web 2.0 Collaboration System for Architectural Decision Engineering. In *SEKE*, pages 255–260. Knowledge Systems Institute Graduate School, 2007.

[40] M. Shahin, P. Liang, and M. Khayyambashi. Architectural design decision: Existing models and tools. In *Software Architecture, 2009 European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*, pages 293–296, Sept 2009.

[41] M. Shahin, P. Liang, and M. R. Khayyambashi. Improving Understandability of Architecture Design Through Visualization of Architectural Design Decision. In *Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge*, SHARK'10, pages 88–95, New York, NY, USA, 2010. ACM.

[42] M. Shahin, P. Liang, and Z. Li. Architectural Design Decision Visualization for Architecture Design: Preliminary Results of A Controlled Experiment. In *Proceedings of the 1st Workshop on Traceability, Dependencies and Software Architecture (TDSA)*, pages 5–12. ACM, 2011.

[43] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 3(52), 1965.

[44] Siemens AG, Politecnico di Milano, Telcordia, and TU Vienna. D3.2 Architecture for Role-Based Governance of Virtual Service Platforms. Technical report, INDENICA Project, February 2012.

[45] K. Smolander and T. Päivärinta. Describing and Communicating Software Architecture in Practice: Observations on Stakeholders and Rationale. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering*, CAiSE'02, pages 117–133, London, UK, 2002. Springer.

[46] R. M. Smullyan. *First-order logic*, volume 21968. Springer, 1968.

[47] M. Strembeck. Scenario-driven Role Engineering. *IEEE Security & Privacy*, 8(1), January/February 2010.

[48] M. Strembeck and J. Mendling. Generic algorithms for consistency checking of mutual-exclusion and binding constraints in a business process context. In *Proceedings of the 18th International Conference on Cooperative Information Systems (CoopIS)*, pages 204–221, 2010.

[49] M. Strembeck and J. Mendling. Modeling Process-related RBAC Models with Extended UML Activity Models. *Information and Software Technology*, 53(5):456–483, May 2011.

[50] K. Tan, J. Crampton, and C. A. Gunter. The consistency of task-based authorization constraints in workflow systems. In *17th IEEE workshop on Computer Security Foundations (WCSF)*, pages 155–169, 2004.

[51] R. C. Team et al. *R: A language and environment for statistical computing*, 2005.

[52] The Open Group. TOGAF 9 - The Open Group Architecture Framework Version 9, 2009.

[53] D. Tofan and M. Galster. Capturing and making architectural decisions: An open source online tool. In *Proceedings of the 2014 European Conference on Software Architecture Workshops*, ECSAW'14, pages 33:1–33:4, New York, NY, USA, 2014. ACM.

[54] D. Tofan, M. Galster, and P. Avgeriou. Difficulty of Architectural Decisions – A Survey with Professional Architects. In *7th European Conference on Software Architecture*, ECSA'13, pages 192–199, 2013.

[55] D. Tofan, M. Galster, P. Avgeriou, and W. Schuitema. Past and future of software architectural decisions – A systematic mapping study. *Information and Software Technology*, 56(8):850–872, 2014.

[56] J. Tyree and A. Akerman. Architecture Decisions: Demystifying Architecture. *IEEE Software*, 22(2):19–27, 2005.

[57] A. van Deursen, A. Mesbah, B. Cornelissen, A. Zaidman, M. Pinzger, and A. Guzzi. Adinda: A knowledgeable, browser-based ide. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE'10, pages 203–206, New York, NY, USA, 2010. ACM.

[58] U. van Heesch, P. Avgeriou, and A. Tang. Does decision documentation help junior designers rationalize their decisions? A comparative multiple-case study. *Journal of Systems and Software*, 86(6):1545–1565, 2013.

[59] U. van Heesch, P. Avgeriou, U. Zdun, and N. Harrison. The supportive effect of patterns in architecture decision recovery – A controlled experiment. *Science of Computer Programming*, 77(5):551–576, 2012.

[60] J. Whitehead. Collaboration in software engineering: A roadmap. In *2007 Future of Software Engineering*, FOSE'07, pages 214–225, Washington, DC, USA, 2007. IEEE Computer Society.

[61] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction.* Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[62] C. Wolter, M. Menzel, A. Schaad, P. Miseldine, and C. Meinel. Model-driven business process security requirement specification. *Journal of Systems Architecture*, 55:211–223, 2009.

[63] C. Wolter, A. Schaad, and C. Meinel. Task-based entailment constraints for basic workflow patterns. In *13th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 51–60. ACM, June 2008.

[64] U. Zdun and M. Strembeck. Reusable Architectural Decisions for DSL Design: Foundational Decisions in DSL Development. In *Proceedings of 14th European Conference on Pattern Languages of Programs (EuroPLoP)*, pages 1–37, Irsee, Germany, July 2009.

[65] O. Zimmermann. Architectural Decisions as Reusable Design Assets. *IEEE Software*, 28(1):64–69, 2011.

[66] O. Zimmermann, T. Gschwind, J. Küster, F. Leymann, and N. Schuster. Reusable Architectural Decision Models for Enterprise Application Development. In *3rd International Conference on Quality of Software Architectures (QoSA)*, pages 15–32. Springer, 2007.

[67] O. Zimmermann, J. Koehler, and L. Frank. Architectural Decision Models as Micro-Methodology for Service-Oriented Analysis and Design. In D. Lübke, editor, *Proceedings of the Workshop on Software Engineering Methods for Service-oriented Architecture 2007 (SEMSOA 2007), Hannover, Germany, online CEUR-WS.org/Vol-244/*, pages 46–60, May 2007.

[68] O. Zimmermann, J. Koehler, F. Leymann, R. Polley, and N. Schuster. Managing architectural decision models with dependency relations, integrity constraints, and production rules. *Journal of Systems and Software*, 82(8):1249–1267, Aug. 2009.

[69] O. Zimmermann, U. Zdun, T. Gschwind, and F. Leymann. Combining Pattern Languages and Reusable Architectural Decision Models into a Comprehensive and Comprehensible Design Method. In *7th Working IEEE/IFIP Conference on Software Architecture (WICSA), Vancouver, BC, Canada*, pages 157–166. IEEE Computer Society, 2008.

Curriculum Vitæ

Personal Information

Name	Patrick Gaubatz
Email	patrick.gaubatz@univie.ac.at
Web	http://cs.univie.ac.at/patrick.gaubatz

Education

10/2010–01/2015	Ph.D. in Business Administration and Economics, University of Vienna
10/2005–10/2010	MSc. in Business Administration, Vienna University of Economics and Business

Work Experience

10/2014–ongoing	Lecturer, Faculty of Computer Science, University of Vienna
11/2013–10/2014	Researcher in the "Field Force" project
10/2010–10/2014	Research Assistant, Software Architecture Group, Faculty of Computer Science, University of Vienna

Professional Activities

Talks

- 29th Symposium On Applied Computing (2014)
- 13th International Conference on Web Engineering (2013)
- 28th Symposium On Applied Computing (2013)
- 4th International Workshop on Lightweight Integration on the Web (2012)

Reviewer/PC Member (Excerpt)

- IEEE Transactions on Services Computing
- Information and Software Technology
- Journal of Systems and Software
- International Workshop on DSL Architecting & DSL-based Architectures
- Working IEEE/IFIP Conference on Software Architecture
- Symposium On Applied Computing
- IEEE International Conference on Service Oriented Computing & Applications
- International Conference on Software Reuse

Teaching

- Information Systems Technology, University of Vienna
- Software Architecture, University of Vienna