universität
wien

MASTERARBEIT

Titel der Masterarbeit

„Managing Responsibilities and Permissions for Process-Aware Information Systems“

varfasst von

Oleksandr Kyshynevskyi

angestrebter akademischer Grad

Diplom-Ingenieur (DI)

Wien, 2013

Studienkennzahl lt. Studeinblatt:	A 066 926
Studienrichtung lt. Studienblatt:	Masterstudium Wirtschaftsinformatik
Betreut von:	Univ.-Prof. Dipl.-Math. Dr. Stefanie Rinderle-Ma

Abstract

Over the last decades business processes and their execution became a central aspect of businesses. When enacting business processes in large scale organizations, also security concerns are very important. Growing businesses engage greater number of participating actors and artifacts in their business processes. Therefore the issue of flexible management of often complex responsibilities and permissions becomes very important. In this thesis we elaborate a means to manage business process related security policies separately from business processes in a modular way. We analyze the previously developed SPRINT framework [16] and further develop an editor which is the basis for implementing the three levels described in [16]: task patterns, responsibility bundles, restrictions. Consequently we elaborate two detailed use cases by means of which the correctness and practical value of the application is evaluated.

Kurzfassung

In den letzten Jahrzehnten haben sich immer mehr Organisation auf die genaue Definition ihre Geschäftsprozessen und deren Ausführung konzentriert. Daraus resultierend gewinnt in weiterer Folge auch der Aspekt der Sicherheit bei der Durchführung von Geschäftsprozessen an Wichtigkeit. Moderne Geschäftsprozesse werden komplexer, dementsprechend steigt auch die Anzahl von Akteuren und Artefakten in diesen Prozessen. Aus diesem Grund steht die Frage des flexiblen Management von Zuständigkeiten und Berechtigungen im Raum. Diese Masterarbeit beschäftigt sich mit der Umsetzung einer Methode zur modularen Definition und Verwaltung von solchen Zuständigkeiten und Berechtigungen, getrennt von den Geschäftsprozessen selbst. Dazu bauen wir auf den SPRINT-Ansatz [16] auf und realisieren einen Editor zur Umsetzung der drei Hauptaspekte von SPRINT: Task-Abhängigkeiten, Zuständigkeiten und Berechtigungen. In weiterer Folge wird die Tauglichkeit der entwickelten Anwendung anhand von zwei detaillierten Use Cases überprüft.

Lebenslauf

Praxiserfahrung

07/06 – 08/06	Praktikum bei Zaporozhjer Kernkraftwerk, Energodar (Ukraine) Assistent des Ingenieur-Programmierers www.npp.zp.ua
02/07 – 03/07	Praktikum bei Zaporozhjer Kernkraftwerk, Energodar (Ukraine) Assistent des Ingenieur-Programmierers
10/07 – 06/08	Programmierer an der nationalen Universität für nukleare Energetik und Industrie
03/08 – 04/08	Programmier-Praktikum an der nationalen Universität für nukleare Energetik und Industrie

Ausbildung

02/2013	1Z0-851 Java Standard Edition 6 Programmer Certified Professional Exam
Ab 03/2010	Magisterstudium Wirtschaftsinformatik an der Universität Wien
10/2009 – 02/2010	Magisterstudium Wirtschaftsinformatik an der TU Wien
10/06 – 05/07	Ausbildung zum technischen Übersetzer (Englisch) Abschlussnote: Sehr Gut
09/03 – 06/08	Studium an der nationalen Universität für nukleare Energetik und Industrie Bachelordiplom als Ingenieur-Systemtechniker Abschlussnote: Gut Spezialistdiplom als Ingenieur-Systemtechniker Abschlussnote: Gut

05/2003	3. Platz beim ukrainischen Wettbewerb für Betriebswirtschaft und Geographie
09/92 – 06/03	Allgemeinbildende Schule, Kalanchack (Ukraine) Abschlussnote: Sehr Gut

Auslandsaufenthalte

09/2008 – 08/2009	AU-Pair in Liederbach am Taunus, Deutschland

Kenntnisse und Fähigkeiten

Fremdsprachen:	Russisch: 1. Muttersprache Ukrainisch: 2. Muttersprache Englisch: fließend in Wort und Schrift Deutsch: fließend in Wort und Schrift Italienisch: Basiskenntnisse
EDV-Kenntnisse:	Java (OCPJP6), Java EE, JavaScript, HTML, CSS, PHP, MySQL, Linux, MS-Office, C++.

Interessen

Programmierung, Technik, Fremdsprachen, Foto, Sport, Musik, Reisen

Oleksandr Kyshynevskyi

Wien, 09.12.2013

Acknowledgments I would like to thank my parents Yuriy and Elena Kyshynevskyi and my girlfriend Tanya for their inspiration and support. I would also like to thank my supervisor Stefanie Rinderle-Ma, Jürgen Mangler for his incredible support and guidance throughout this thesis as well as Maria Leitner for help with correction and the use cases.

Contents

1 Introduction

1.1 Motivation

With ongoing growth and increasing complexity of organizations as well as increasing complexity of regulatory frameworks business processes get more and more complex involving hundreds or even thousands of actors [17]. There are several major drivers explaining this phenomenon. The first one is the information technology (IT), which enables businesses and non-commercial organizations to run more effectively, opening new spheres of activities, unknown before. IT makes it possible to develop new and sophisticated business models (Google, Facebook, Amazon) thus giving us, the consumers, additional value.

On the other hand the ubiquitous globaliazation of economies, growing competition in all spheres of business forces companies to be more cooperative than ever. For example, a virtual enterprise can be considered as an extreme form of cooperation between two and more (commercial) entities: if one

company meets a kind of challenge, which it cannot handle by itself (due to insufficient production capacities, resources, missing levels in the production chain, know-hows, etc), it can partially be entrusted with external partner(s). This process does not demand a new legal entity to be created, but conversely a new expanded business process should be designed. Having achieved their goals such virtual enterprises are normally subject to deresolution afterwards, letting its participants seek new possibilities in the dynamic business world [8].

Organizations can also cooperate in a more permanent way, which also (as in case with virtual enterprise) requires that some production stages should be performed by geographically dispersed partners. Finally, the structure of one organization can be complex enough to include dozens of divisions and coworkers who share same production inventory and artifacts. One can think about a hospital, where a surgical ward is getting prepared to perform a long and complex operation, demanding a series of various analyses and procedures to be made in a pre-stage. This entire process involves a number of actors with specific roles and rights. The information generated during preparation should flow to the right actors, restricting access to those, who aren't meant to get it.

Systems that help with an enactment of business processes, are called Process Aware Information Systems (PAIS). The enactment of a business process through PAIS typically requires the process to describe control flow, data flow and users or services, cleared to execute this Business Process (BP). It can also include some access-sensitive data, which only specific actors or services are allowed to use [16]. This task in PAIS is solved by means of security policies [3], containing access rules and authorization constraints. This restrictions can be furthermore tightened by context policies [10], determining time, data or location, from which a BP can be executed.

Till now there were several approaches of managing security policies, each of which having its own drawbacks or limitations. The main problem is the partitioning of security functions: for example, a policy may contain access rules and authorization constraints without having context specific rules. In other case access rules may be integrated in a BP, leading to expanding of this process and making it less cohesive. In that way, we can see the dispersion of various security aspects over the PAIS instead of managing and storing them centralized.

The goal of this thesis is to implement a system, which will allow to create, maintain and enhance security policies in a way, as promoted by Security in PRocess-aware INformation sysTems (SPRINT), which implies separation of security policies and process logic and representation of policies by means of responsibilities, permissions and constraints. In a future this could possibly allow to solve the problem of numerous process-aware information systems, each representing security concepts its own way.

1.2 Contribution

This master thesis contributes to previous works in the field of security management in information systems. Its main starting point is the paper [16], whose goal is to develop an approach for managing the security aspects in information systems. Another foundation is OrgViz [12] - previously developed application for representation of enterprise organizational structure. The main goal of this thesis is to build a prototype, which helps security engineers to handle the security aspects in business processes.

The following is considered to be the contribution in this work:

- implementation sketch: the concepts from [16] are reviewed and customized in order to be implemented in a web application, that allows security engineers to easily define and maintain security related rules for business processes;

- three editor components and a backend server. Analysis and sketching lead to a web application aimed at composing (1) task and (2) responsibility patterns as well as (3) constraint editor. Each

of the tree editors yields a pattern, whose structure is defined by the application. These resulting patterns are stored on the backend server;

- application of the SPRINT [16] for two detailed use cases (reactor maintenance and university enrollment) in order to test the functionality of the program. As it turned out, contemplating about how to implement certain constraints led us to a new way of using unary operators in the pattern rules, which we will see in the section 5.

1.3 Methodology

In order to realize the goals of this thesis, we employed an approach roughly based on the theory of Action research [2]. We analyzed the input provided by the SPRINT approach, designed an improved system and evaluated it according two to use-cases. Our concrete steps, mapped to chapters in this thesis were:

- Sect. 2 and 3: Analysis of existing approaches, which regulate security aspects of information systems.
- Sect. 4.1 Elicitation of use cases to later check the appropriateness of the three editors;
- Sect. 4.2 and 4.3: Design and implementation of prototype;
- Sect. 5: Evaluation on the basis of the prototype and the uses cases.

1.4 Structure of Thesis

In Sect. 2 we discuss some important security models in information systems, which will guide us into the topic. Sect. 3 gives detailed description of the SPRINT approach, which in turn uses Role Based Access Control (RBAC) in order to describe the concept of security management in large scale systems. In Sect. 4 we present the use cases which we will later implement using the target application. This section also shows some technical aspects about how the SPRINT approach may be utilized in a real-world application. Sect. 5 checks the suitability of the application by means of carrying over the use cases from Sect. 4 into the program. Sect 6 wraps up the work and summarizes the possible future steps toward building robust and reliable framework for managing security aspect in a large scale information systems.

1.5 Glossary of Terms

access control the set of means aimed at limiting the availability of some resources for unauthorized persons (also services)..

access restrictions physical or informational barriers which protect resources from unauthorized access..

BP Business Process.

business model a description of process of how an enterprise delivers a product or service for consumers..

business process a set of interconnected activities which lead to a creation of product or service..

NIST National Institute for Standards and Technology.

RBAC Role Based Access Control.

security policy a set of rules with security requirements for the given system..

SOD Separation of Duties.

SPRINT Security in PRocess-aware INformation sysTems.

workflow may be seen as an instance of business process. Involves data objects and actors which perform tasks from BP.

2 Access Control in PAIS/Background

The organizational security has been an issue for quite a while. There were different approaches to standardize this area of interest in this work we will use concepts of resource-based access control, developed by National Institute for Standards and Technology (NIST).

2.1 NIST-RBAC

The **NIST-RBAC** Model stands for National Institute of Standards and Technology Role Based Access Control. It is a security standard that was developed as an extension of existing RBAC. We first look at original RBAC, its weaknesses and drawbacks and after that go through improvements made by NIST [23] [7].

Role based access control is an established approach for authorization in a large-scale systems. It uses the concept of roles, which was known long before the first computers have appeared, and was widely spread in the privilege controlling realm. However because of lack of standards, the concept of roles was implemented in different ways, as various vendors had their own vision of what the role is and how it should be squeezed into a system.

In its nature the RBAC principle is rather abstract, as it doesn't rely on a user identity, which is entrusted to perform specific actions on the data artifacts or in some system in general. The main concept here is a role which has specific permissions attached to it. After defining the roles in the enterprise, they must be assigned to employees. One employee may have one or more roles. A role can imply permissions, skills, responsibility of the coworker. As an enterprise could possibly present enormous structure with hundreds, thousands or tens of thousands of employees, RBAC is a good way of maintaining security prescriptions: if a company hires a new coworker, there just should be an existing role assigned to him. Furthermore large companies may have partially or entirely overlapping role permissions. RBAC proposes here a hierarchy approach, which implies that higher standing roles include permissions of their sub roles.

Role based access control also allows for specification and enforcement of security policies which can be specifically developed for individual enterprises. If enforced with a particular system, this is result of configuration of various components of [23]. That makes a great difference to classical discretionary and mandatory access control systems, with policies embedded into access control model. As RBAC supports the separation of duty constraints between roles, it makes possible to provide additional mechanisms for specification and enforcement of policies.

Just because role based access control concept showed its efficiency, it was implemented by many software companies in their products where access restrictions play significant roles (DBMS, operating

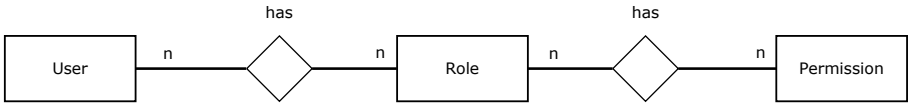


Figure 1: Flat RBAC by [23]

systems and others). But because of missing generally accepted standard, each company tried to define RBAC features its own way. As a result a set of heterogeneous RBAC models is ensued. That explains the attempt of NIST to standardize this approach.

It was admitted that the RBAC concept can vary from very simple to considerably complex forms. That makes a single universal model of RBAC not quite acceptable. An advanced model would contain a lot of overhead for most use cases. Therefore the NIST model consists of four subsets, each requiring the previous one to be implemented. Each level brings additional functional features into the model.

Flat RBAC is an initial level of RBAC. The essence of this level is that each user obtains a role. Each role is assigned to a set of permissions. The user-to-role and permission-to-role assignment can be many-to-many. Each user can also exercise permissions from multiple roles. Thus flat RBAC embodies features of traditional group access control.

Hierarchical RBAC takes a step further and introduces the principle of role hierarchies, where higher role includes permissions of the subordinate role. There are two sub-levels to distinguish between:

- General hierarchical RBAC – supports arbitrary partial order;
- Restricted hierarchical RBAC – support for possible restriction in the role hierarchy tree.

Hierarchy distribution can be inheritance-based (upper roles obtain sub-role permissions) activation-based (excludes permission inheritance) or mixture of two above. Due to its value this feature is widely implemented in various products.

Constrained RBAC supports Separation of Duties (SOD), which are used to identify and avoid conflict of interest policies such as to prevent users from fraudulent actions. SOD should meet static SOD (user-role assignment) and dynamic SOD (role activation) requirements. This feature widely finds its application in sophisticated systems involving a great deal of actors.

Symmetric RBAC allows for permission-role review in a way, flat RBAC allows for user-role review. That means, we get a possibility to determine the roles accommodating given permission. In efficiency terms it delivers about the same performance, as the flat RBAC.

Fig. 1 shows the first level, which regulates the group-based access to some objects. This model level is a starting point for the rest three. A great advantage of the flat RBAC is that it supports many-to-many. Most of the modern systems vary in respect to the number of roles that particular user can obtain. This number reaches from 16 or 32 up to hundreds and even thousands. Though the NIST model for this level of RBAC doesn't regulate the minimum for the roles quantity, this point should be handled by concrete operational environment.

In Fig. 1 the User concept is context specific and can represent a human being or some software unit, able to perform actions (e.g. web service). A Role may be the function in an organizational structure or any other functional bundle. Lastly the Permission concept determines a set of rights to treat objects one or another way. NIST-RBAC determines only positive permissions as opposite to negative permissions which forbid actions on some objects. Furthermore concrete system may support sessions. In that way user obtains specific rights only with begin of new session. During each session a user may explicitly indicate, which set of permissions is necessary at this moment of time. Linux users are well familiar

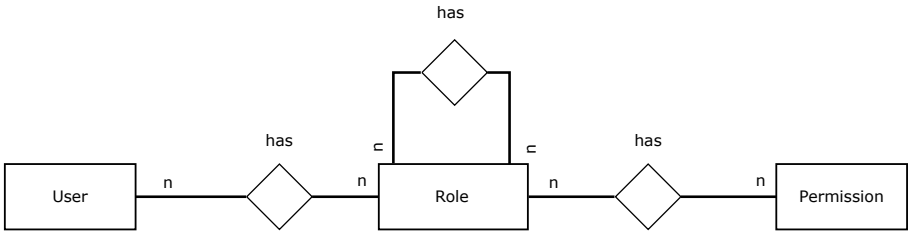


Figure 2: Hierarchical RBAC by [23]

with this mechanism: normally a person logs in as unprivileged user to prevent occasional damage to the system. Only when special privileges are required, root-session takes place. A significant RBAC requirement is that multiple roles can be activated simultaneously.

Flat RBAC enables querying to determine which users hold specific roles and vice versa. The same is true for role-permission relations. Having specified main functionality points, flat RBAC model leaves some mechanisms for software vendors to consider:

- How should scalability work in the system.
- The actual implementation of revocation/activation. This happens, if a user is subject to removal from the system or if some if role's permissions should be shrunk.
- Who should assign roles to users and permissions to roles.

These omissions are made intentionally to let vendors adapt their products for specific market needs and to let them implement these features in a way that is most appropriate in a given situation.

Hierarchical RBAC model (Fig. 2) differs from flat RBAC only by role hierarchy relation. Role hierarchy represents the internal order of duties and rights. In the most common cases senior or more powerful roles are put at the top or hierarchical tree with less extensive roles descending from them. The principle of arbitrary partial order leads to a two sub levels distinction:

- general hierarchical RBAC with a support for arbitrary partial order;
- Limited hierarchical RBAC with possible restriction for a structure of hierarchy tree, which in turn can be normal or inverted tree.

The tree structure of a hierarchy is well suited to represent inheritance mechanisms for role permissions. Fig. 3 shows examples of hierarchies with junior-most role (bottom) and without it (top).

Both hierarchies however include senior-most role (DIR). That is considered to be dangerous, as this role derives too many permissions. Even if a person entrusted with these privileges is considered to be exceptionally reliable, it may commit a mistake, leading to negative consequences. To make the entire system less error-prone some procedures may be engaged that limit the permission inheritance. Similar to object-oriented programming languages, those roles, permissions of which are not inherited, are named private (just like private instance variables and methods are invisible for other classes and subclasses in a class inheritance tree).

In regard to hierarchies two interpretations exist to distinguish between: we're already familiar with the first one, stating that senior roles inherit permission from junior ones. This interpretation is referred to as permission-inheritance interpretation. The activation interpretation on the other hand prescribes

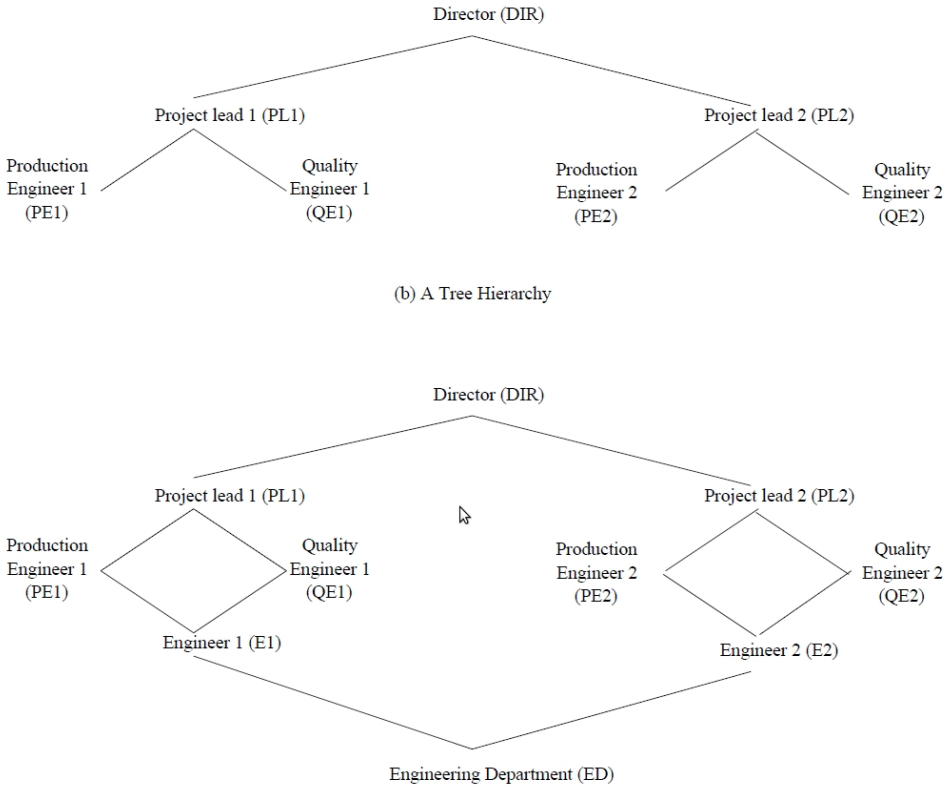


Figure 3: RBAC: Hierarchy Example by [23]

that activation of the senior role does not mean the activation of the junior roles'permissions. In order junior roles to obtain their permissions, explicit activation is needed. Modern software systems may apply both methods simultaneously. Another principle which should be mentioned is the Access Control List approach (ACL). It is widely applied in file systems and for each file system entry defines the list of processes and users as well as concrete operations, which are allowed to be performed on that entry. The difference of the ACL from the two above approaches that there is no permission inheritance. In this respect the ACL may be compared to a so-called "minimal RBAC Model".

Constrained RBAC adds constraint features to a preceding hierarchical model. The meaning of constraints is to limit the user-role assignment or to regulate the role-permissions activation during a session and thus to implement mechanisms for separation of duties. This will prevent user to go beyond the prescribed permissions. That also eliminates a possibility to consider, a fraud or malicious action could stay untracked in the system. Least privilege principle finds its application here. In order to perform some action inside the system, a user gets exactly the minimal amount of authority to fulfill this action. That reduces potential risks of damage.

We now take a closer look at static and dynamic SOD between mutually exclusive roles that claim to be another one effective technique. In order to prevent a user from obtaining permissions of two conflicting roles a static SOD can be activated. That means if a person acting on behalf of a user in the system and being a member of role R1 wants to perform next step as being member of role R2 this step will be blocked or allowed only after permissions from R1 are given up and user logs in as a member of R2 (provided, he has a possibility to do so). Speaking in inheritance tree terms: if roles R1 and R2 are constrained by means of SOD the R1's higher role inherits this constrain with respect to R2. That

implies no SOD is applicable between R1 and its higher role because that would deny the core principles of inheritance. In order not to get away from Linux/Unix theme, SOD may be seen in action when changing users with *su - user2* command in terminal. That allows user (or users) to change the roles from current *role1* to *role2* and to obtain different sets of permissions. In order to get back and become *user1* again should use *su - user1* anew.

The dynamic separation of duties is pretty similar to static SOD, but there is one crucial point that makes difference: dynamic SOD limits the availability of permissions within or across a users session [7]. Assume a user is allowed to act on behalf of normal bank clerk and a clerk supervisor. In a situation when an action performed by clerk must be submitted for a supervisor approval, this user might want to log in as a clerk supervisor to make approval without resigning its authority as a normal clerk. At this point a system will deny access as long as the user keeps the clerk role. Till now there is no actual distinction to static SOD with one exception dynamic SOD provides more flexibility. But here is where things get different: the previous example makes it clear that R1 and its higher role belong to the same inheritance tree. That was not possible in the static separation of duties. So it is perfectly fine to have multiple roles connected through inheritance until a user undertakes effort to obtain permissions for more than one role simultaneously.

Though the principle of separation of duties may be implemented by sole use of flat RBAC, the NIST model requires the possibility to build hierarchical role structures, as this eliminates limitations for flexibility and functionality.

Symmetric RBAC adds the concept of more complex permissions to the preceding levels of RBAC. These permissions concern needs for effective reviewing mechanisms in the systems. All this may appear to be redundant, but modern enterprise may comprise a geographically dispersed entity with several branches. That means a system administrator should be able to do role-responsibility and user-role revisions for all enterprise parts regardless of the geographical location. Furthermore, permissions a specific role is associated with may include objects from different locations. Crucial point here is the principle of the least privilege.

There are more than enough reasons to review the policies in the enterprise. Firstly the user may leave the company (fired, suspended, etc.). In this case all its privileges and roles should just be permanently or temporary taken away. We can often hear news reports about angry employees, who did massive damage to their company after being fired. That was possible solely because system administrators didn't cancel all due permissions. In this case it would be most appropriate to completely remove all the access data associated with particular user, as when left in the system this data may present potential security risks. Secondly a user may get promoted, implying wider range of permissions. Here system administrator should carefully activate additional set of responsibilities.

As it was already mentioned before, starting with RBAC level 1 the interface for reviewing role-user and role-permissions relations are required. Level 2 of NIST-RBAC claims features to effectively monitor and manage not only permissions, which were directly assigned to specific role, but also those, which were inherited from the sub roles. As we proceed to RBAC level 4, there should be a feature to provide results for two types of requests:

- a request to return a complete set of objects associated with some permission;
- a request to return a complete set of the object-operations pairs associated with permission.

Optional feature may also display direct and indirect permission assignment (indirect regards to permissions obtained through inheritance) and a feature to perform a review on a concrete subsystem. Being rich and open ended approach, NIST-RBAC doesn't touch upon some subjects which software vendors are free to interpret in their own way. The most important of them are:

- Scalability is an individual parameter for enterprises. Some businesses rely on products with low scalability indexes (may bear further problems in case of rapid growth), the others prefer more flexible solutions. Scalability may regard the number of roles, permissions, size of role hierarchy, limits on user-role assignment and so on. Generally a further scale is adopted: small systems are extensible up to tens of times, medium up to hundreds and finally large scale systems are capable of 1000 times extension.

- Authentication mechanisms determine the way users log into system and obtain their privileges, attached to some role. This should be the part of system architecture and lies beyond access control model. Negative permissions are also sometimes implemented by vendors and are there to forbid access. RBAC omits them because this concept might bring some confusion and redundancy. The concept of normal (or positive) permissions implies that user has no access to those objects, for which it has no positive permissions.

- The nature of permissions may vary widely and concern for example the granularity of permissions, e.g. fine- vs. coarse-grained (access to single object or a group of objects respectively), some functional perspectives (commit, check), artifact-related operations (create, delete);

- Discretionary role activation: as NIST-RBAC doesn't prescribe mechanisms for how and which roles should be activated in a case, a user has more than one role. Single requirement here is a possibility to activate all roles simultaneously. Various products act differently in this area: some allow for activation of default features and let a user afterwards deactivate/activate un/necessary roles.

- Constraints: as we already have seen, RBAC prescribes two types of constrains (static and dynamic SOD) which belong to prohibition ones. A further example is obligation constraint, whose name is self explaining.

- Role revocation represents a feature of high importance. The main advice here is that as soon as given permissions are no longer needed they should be revoked.

2.2 W-RBAC

W-RBAC stands for Workflow RBAC [26] and introduces mechanisms for role-based access control in workflow systems. W-RBAC is represented by two models: W0-RBAC and W1-RBAC. The first one uses the above described permission approach from RBAC along with workflow constituent separated from each other to make the management of administrations easier. W1-RBAC carries additional features, enabling exception handling during the execution process. Let's take a closer look at both models.

We're already familiar with the RBAC model, which includes such concepts as **User**, **Role** and **Permission**. W0-RBAC adds some additional concepts to it, namely case C and Organizational Unit OU. Consequently the four new relations are coming into play: **include**, **member**, **head** and **doer** (Fig. 4). As most important permission in our case is a task execution permission, we omit some administrative ones, such as adding new task, users, constraints, etc.

As one might remember, the NIST-RBAC model introduces protection mechanisms, which prevent one user from executing discrepant roles at the same time. For example the same user cannot request for credit and approve this request simultaneously. This user however may request credit in one operational case (for example bank may credit their own employees) and approve credit request in the other (normal credit request from non-employee). The key point here is that multiple contradictory roles shouldn't be activated for the same user in one session, but may be legally activated in different sessions.

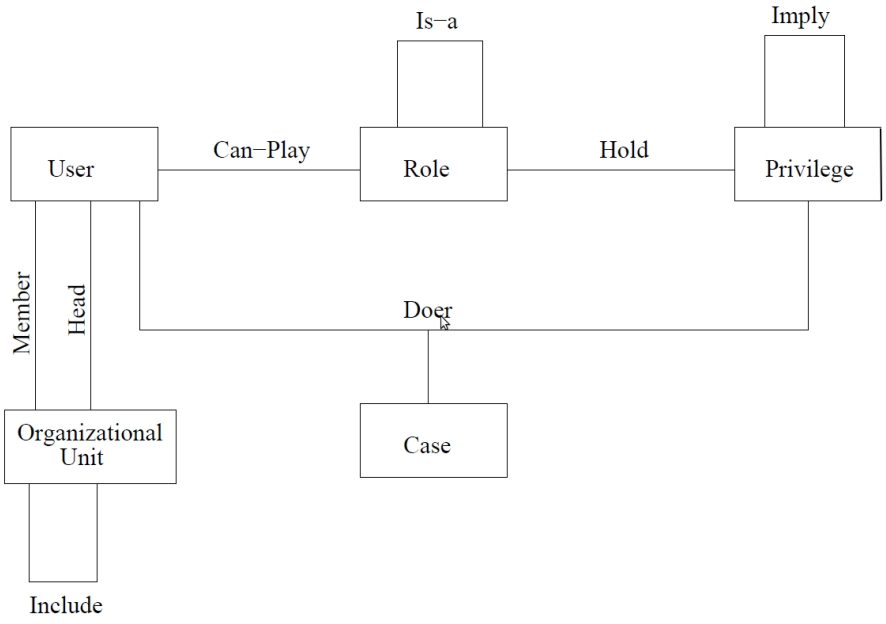


Figure 4: W0-RBAC Meta Model by [26]

The concept of sessions is not that distinguishable in workflow systems, as there are no such distinct time boundaries. It is also critical here, that one user obtains both roles simultaneously as credit requester/approver. That is what **case** concept stands for. From example above we've seen that a bank employee may approve a credit request and may be a requester in different cases. The key point for him is not to be requester and approver in the same case.

In that way, cases are used to represent instances of business processes for workflow systems. The ternary relation **doer(u, p, c)** connects three classes in the meta model: the case, the user and finally permission. This implies, that a particular user obtains certain set of permissions for a certain case.

Next we get to a concept of Organizational units, which by convention serves to represent the hierarchical structure of the enterprise with its employees and heads of departments. Furthermore there can be static (such as typical accounting or HR department) and dynamic units (e.g. temporary project with project team and one or more project lead). But it is important to distinguish between organizational hierarchy and the roles hierarchy that we know from the NIST-RBAC model. Hence the is-a in RBAC is different from the power is-a. In RBAC is-a stands for inheritance of responsibilities rather than subordination. But that is true up to some extent. In some cases it is irresponsible to entrust the highest roles with all the responsibilities of the subordinate ones. Thus the head of the bank should not have the responsibilities of the safe manager who is the only one to know the password. The last three new relations to RBAC model are:

- **include** (d_1, d_2), d_1, $d_2 \in OU$ means that organizational unit d_1 includes organizational unit d_2;
- **member** (u, d), $u \in U$, $d \in OU$ represents the belonging of user u to the organizational unit d.
- **head** (u, d), $u \in U$, $d \in OU$ means that user u is in charge of the organizational unit d.

Constraints are represented in W0-RBAC by means of a standard logic program clause C_n. The predicate added to this constraint reflects a state, when constraint is being violated and a situation described by Cn is invalid $\perp \leftarrow C_n$. Constraints may be expressed as follows:

$$\bot \leftarrow A_1, \ldots, A_k, \text{ not } B_1, \ldots, \text{not } B_l$$

Here either k or l may equal to zero. The above expression may be also represented as $p(t_1, t_2, ..., t_m)$. The predicate p here may be any of the relations from the W0-RBAC meta model or any recursive relation to those from meta model. ti may represent either existentially quantified variables or instances of the concepts from the meta model.

Furthermore, constraints may be static and dynamic. Static constraints are independent of the process instance execution state and define the relations between the concepts of meta model that should not happen by means of specifying certain conditions which may lead to such relationships. Static constraints must be checked after any transaction resulting in adding/removing a tuple, which implies an adding of a new instance of the meta model class.

Constraint may be referred to as dynamic if there are any doer predicates in the clause. The role of dynamic constraints is to prevent a user from performing some action. One can distinguish between following cases:

- Dynamic separation of duties: restricts from performing contradictory activities by the same user. We've seen his type of constraints in NIST-RBAC model. For example:

  $$\bot \leftarrow \text{doer}(u, T_1, c), \text{doer } (u, T_2, c)$$

  means that one and the same user u is not allowed to perform task T_1 and T_2 for the case c.

- Binding of duties is opposite to dynamic SOD: if user u has performed some task T_1 he must perform another task T_2 for the same case c. The said above may be expressed as following:

  $$\bot \leftarrow \text{doer}(u, T_1, c), \text{doer } (u, T_2, c), \text{not } u' = u.$$

- Inter-case constraints are the ones spreading over multiple cases. One of the useful applications may be counting the number of times a user has performed certain task as each task execution may be considered to be a separate case.

- Reciprocal separation of duties prevents form forming coalitions and achieving biased results across cases. For example, if user u_1 approved the request for user u_2, then user u_2 cannot approve any request for u_1.

  $$\bot \leftarrow \text{doer } (u_1, request, c_1), \text{doer } (u_2, approve, c_1),$$
  $$\text{doer } (u_2, request, c_2), \text{doer } (u_1, approve, c_2), \text{not } c_1 = c_2.$$

So far we discussed the W0-RBAC and the constraints definition. There are cases which require decisions to override constraints supposed to be less important. This concept is supported by W1-RBAC. Imagine a situation where a hospital accepts a patient with a serious injury. In this case some formal procedures must be omitted so that the patient gets the treatment he/she needs as fast as possible. Furthermore some tasks restrictions should be violated in a situation if for example no second surgery assistant is available for an operation on this patient (the operation must be conducted without second assistant or a person not qualified as assistant should take his/her place). Of course this kind of situation must be handled with exceptional responsibility and caution, as by involving employees for the tasks they're normally not allowed to perform, the a lack of people to perform their direct tasks might happen. In such way, if we enable unqualified person (say, nurse) to be a second surgery assistant, a situation is possible where no nurse will be available to accept a newly delivered patient.

The described above situation is called *dead end* (no potential executors for a certain task). There is also another reason for such cases: normally W-RBAC systems allocate executers for the tasks during the runtime (i.e. task activation). The process of allocation may be based upon such information as executor availability and work load. To recover from *dead ends* some constraints should be overridden. Let's consider another example: the policy of the hospital requires the treatment of the patient by the same doctor, who has examined the patient. But because this doctor D, who has examined patient P two weeks ago, is unavailable (having vacations in France) the treatment of P (if cannot be postponed till the necessary doctor is available again) should be relayed onto another doctor D_1.

It is worth to mention that not all the constraints have the same degree of importance. The mentioned above hospital example constraint may be referred to as binding constraint. The goal of binding constraints is to provide reliable, effective and unbiased treatment (in some cases also security). W-RBAC model allows for assigning the degrees of importance to constraints, i.e. to prioritize them. We will see how to do that in the next step.

Previously we have defined constraint as

$$\bot \leftarrow C_n$$

Next we assign a number to each integrity constraint rule

$$\bot \leftarrow C_n \text{ priority } 8.$$

The higher level number expresses the higher priority of constraint. Usually the positive integers are used to express the priority. Furthermore we can gather all constraints with the level of i in a bundle C_i , which gives us the level of compliance of formula.

There should also be defined which constraints may be overridden and which users are allowed to override them. As we already know that users obtain their permissions through roles, an attribute **override**(n) may be specified for every role, which means the certain role is capable of overriding the constraints up to level of n. The more powerful roles may override constraints of higher levels. In order to implement the mechanisms to protect some constraints from overriding the priority level i may be assigned the value enough high, so that no roles are able to override this constraint.

2.3 Administrative RBAC

The Administrative RBAC Model (Fig. 5) may be considered to be an extension of the traditional RBAC [22]. In this chapter we will discuss ARBAC97 model. There are newer models (like ARBAC99), but they all follow the same original principle.

ARBAC is represented by three components: URA97 – the user-role assignment, PRA97 stands for permission-role-assignment, RRA97 – role-role assignment. ARBAC is distinguished from traditional RBAC by the presence of administrative roles and permission, which are used to manage (administer) the formal roles and permissions.

Let's first take a look at the first component of the ARBAC97, the URA97. The two main objectives of this component are to grant and to deprive the privileges through participation of specific user in some roles. This mechanism can be also applied for assigning user to groups. Important moment here is to decentralize the role assignment, in other words to make it appropriate for large-scale systems. For example in an extensive system each smaller subsystem may have a junior security officer (JSO), whose authority is limited to administer some bounded set of roles (say, A, B, C). Important here is to secure, that JSO is only allowed to add users to these roles and to restrict to some extent the circle of users, which may be assigned to roles A, B or C. So, the main concern of the URA97 is to determine, which user may be assigned to specific role, who may conduct this assignment and to separate the authority to

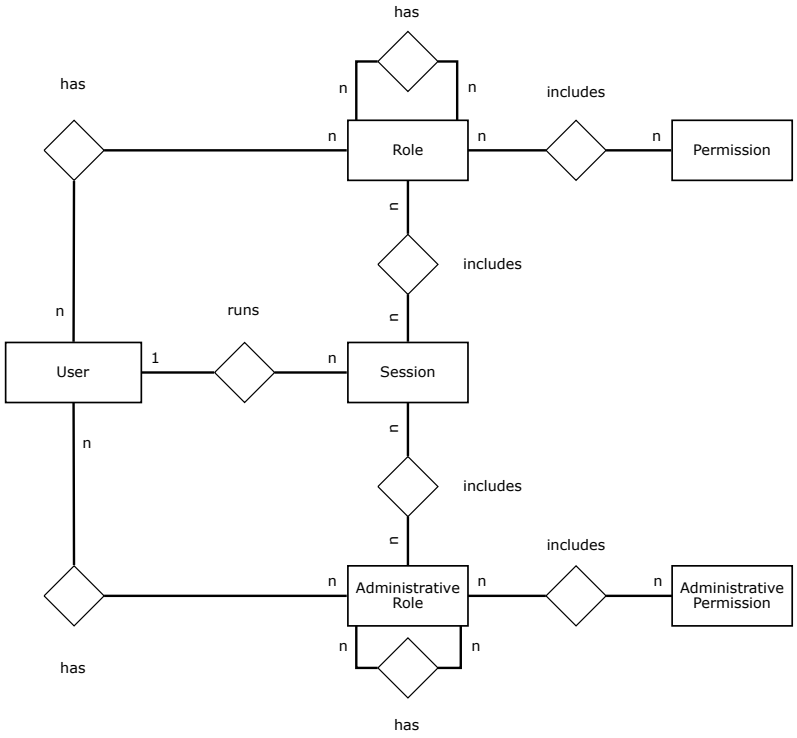


Figure 5: Administrative RBAC by [22]

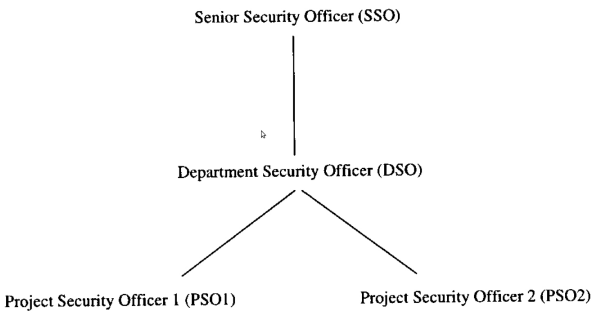


Figure 6: Administrative Role Hierarchy by [22]

add/remove users to/from the role. In that way the URA97 assumes the existence of administrative role hierarchy along with existence of normal role hierarchy. Example of such hierarchy may be seen at Fig. 6.

2.4 Visualization of Organizational Models

While this master thesis is using the principles developed and described in [16], it also relies on a notion of the organization itself, that is similar to WRBAC. the organization is represented by organizational units (divisions, subdivisions, etc) and people who work there. This part is covered in [12] and implemented through the corresponding web application. We will use it later in order to represent the structure

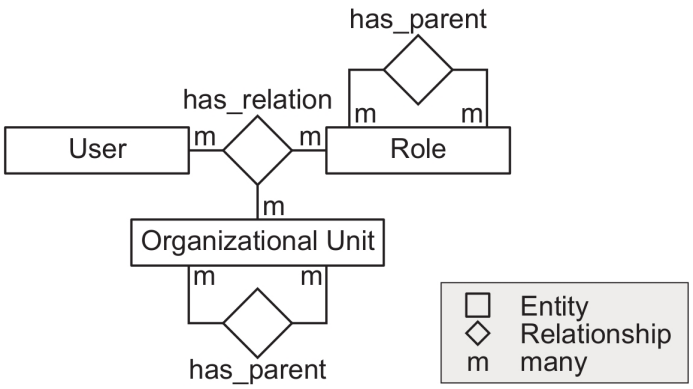


Figure 7: Meta Model for Structure Representation

of organization for the use cases, described in section 4.1.

The main arguments of the authors of [12] is the importance of a user friendly organizational model, which allows to navigate through its functional entities, to query the roles and to be able to obtain the list of employees, who are entrusted with certain set of privileges. As we have already mentioned in section 2, the concept of role is more universal, as the staff pool is more likely to change over the time: employees may enter or leave organization, get promoted, etc [1]. The whole situation is likely to get more complicated with larger organizations, represented by tens of organizational units and thousands of employees. Without proper ahndling, changes in organizational structure might yield some serious security breaches [27] [20] [21]. That is why there is strong need for robust but efficient model [15] [13]. Thus having a good model and visualization allows to avoid security problems, by hiding the complexity.

[12] proposes two ways of representing organizational structure - the *OrbitFlower* and *OrbitList* (Fig. 8), both of which are represented by set of nodes and edges, connecting those nodes with each other [24]. The *OrbitList* puts the structure hierarchically [9], whereas the *OrbitFlower* stands for the network visualization, which in some cases may be more preferable [4]. Both approaches have their advantages and disadvantages, but the ability to switch from one view to another for the same target organization makes them very versatile. It is important to notice, that the meta model for both approaches includes ternary relation between user, role and organizational unit (Fig. 7). This point is crucial, because such approach allows to makes queries in different ways, thus retrieving much more information from the model, than it would be possible for traditional hierarchical approach with binary relationships.

We now explore the meta model in Fig. 7 and its semantic in more detail:

- The Role's entity *has_parent* binary relation stand for classic hierarchical representation of the organization's structure.
- The same holds true for the *has_parent* many-to-many relation for Organizational Unit.
- The User entity has relation to Organizational Unit and Role entities, which mean a user with certain role may belong to 1..m organizational units or a user in organizational unit may have 1..m roles.
- The Organizational Unit entity is comprised by 1..m roles and 1..m users;
- A Role of one particular user may occur in 1..m organizational units.

The concrete users in Fig. 8 are listed separately from Organizational Unit - Role diagram. This is made for the reason we mentioned above, as the internal staff rotation is subject to frequent changes.

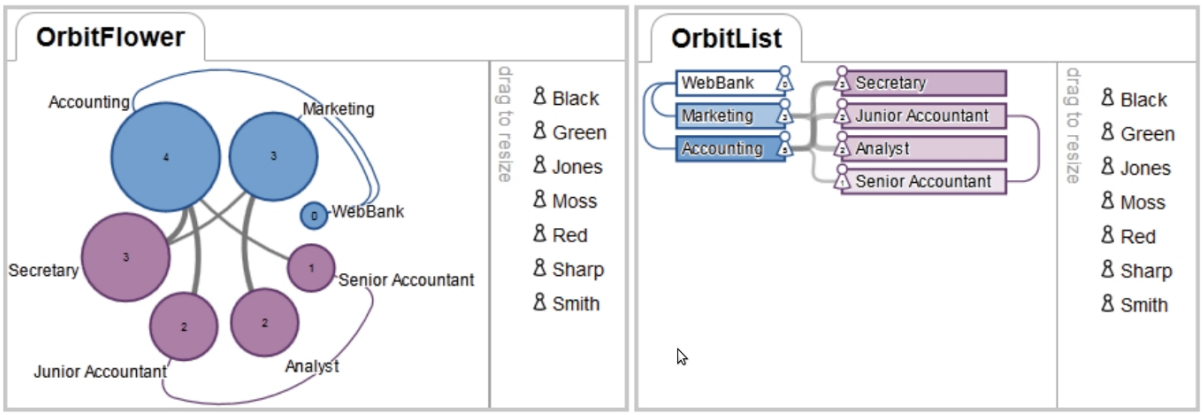


Figure 8: Example of the Orbit Flower and List

Furthermore, with larger organizations it would be problematic to display all available users, which may turn the entire diagram to cluttered and unreadable. Instead the interface of this application allows to filter the list of users in organizational unit or users, entitled with specific role, as the mouse cursor moves over (or clicks) specific unit or role respectively.

In order to make the distinction between organizational units and roles more clear, different colors are applied: the blue for the first and purple for the second entities. The thickness of the edges, connecting organizational units with corresponding roles varies depending on the number of users assigned. Lastly we point out the difference between representation of nodes in both models:

- **OrbitFlower**: the size of circles standing for organizational unit/role depends on the number of users assigned to corresponding entity, whose exact number is specified in the middle of the circle.
- **OrbitList** uses a bit different way to express this information: instead of adjusting the size of the node, the color intensity is used there. In order to specify the number of users in the entity, the user silhouette with number on it is used.

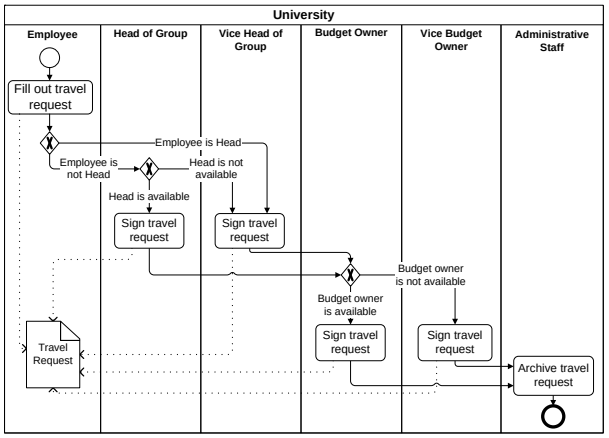
3 Conceptual Design – the SPRINT Approach

We are now discussing the basic approach for managing security aspects, which we rely on throughout this thesis. Principles from SPRINT will be used in order to implement security aspects in our target application. This section provides a summary of [16].

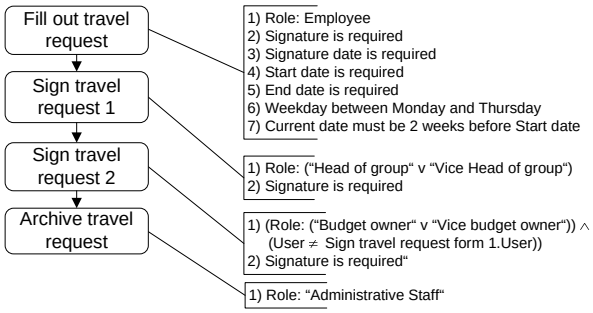
3.1 Structural Aspect

As already discussed in the introduction, there are different ways to include security aspects in the process [25] [5] [18] [14]:

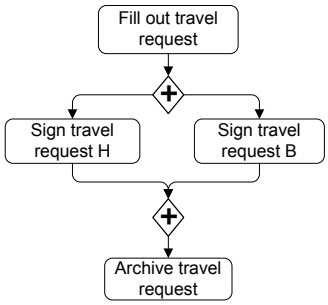
- as a part of a process
- attached to concrete process;
- as a common set of prescriptions, completely decoupled from the process, but from which a prescription for every single step in the BP can be derived (exact understanding of the entire process is required);



(a) Inherent Representation



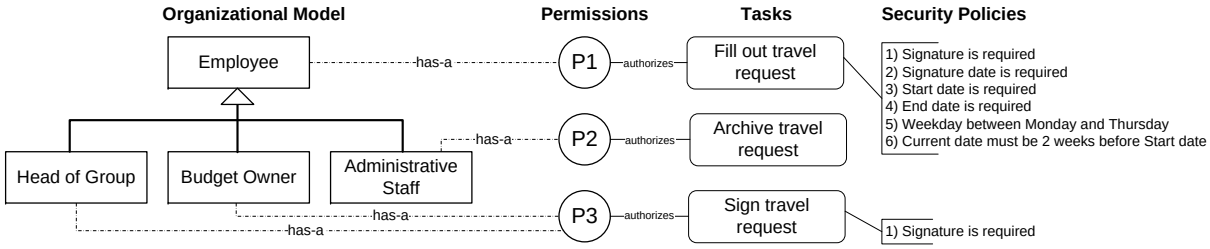
(b) Attached Representation



Security Policies

- An employee has to sign the request before a head or budget owner can sign the request.
- An employee has to file a request two weeks before the traveling starts.
- A travel req. can only be filed between Mondays and Thursdays (for accounting reasons).
- A travel request applies only to employees of a specific faculty (e.g., A,B,C) at University. (Other faculties at University may have different procedures.)
- A travel req. has to be signed by a head of group and a budget owner.
- A travel req. has to be approved by two different persons.
- If an employee is the head of group, then a vice head has to authorize the travel request.
- In case a head of group is not available a vice head is authorized to sign the request.
- If a budget owner (employee might be budget owner) is not available, then a vice budget owner is allowed to sign the request.

(c) Separated Representation



(d) Task-based Representation

Figure 9: Travel Request: Process Modeling and Security Policies by [16]

- as permission set to perform certain action. Each permission is associated with the concrete role from the organizational hierarchical tree.

Figure 9 demonstrates the above approaches in more detail.

Since in a first approach security aspects are incorporated in the business process directly, thus being a part of BP, only the last three techniques may be referred to as supporting security policies. This corresponds to a general opinion that policies should be decoupled from the process and managed/stored separately.

Being RBAC-aligned SPRINT concentrates on the last approach of four. The main goal here is for system security to be separated from concrete business models and modeling notations, therefore SPRINT represent the four main requirements:

1. **Independence** of security policies: in this way the BP model and policies may be designed apart from each other.

2. **Maintainability** of security policies claims for an ease of typical maintenance operation (change, delete, create). The ability to reuse policy in multiple business process models is also covered here.

3. **Extendibility** of security policies: with respect to dynamic nature of the modern world there should be no need to know security policy when changing business process model, as well as no understanding of the business process should be required in maintenance of policies. On the other hand there should be a mechanism providing warnings in case of violation of security policies.

4. **Scalability** of PAIS components refers to the ability to manage enlargement of business processes with respect to security issues. In order to achieve decent scalability the process should be properly and efficiently designed.

   There are several techniques helping to acquire security policies. They vary form process mining to direct interviewing of the process participants. In the second case employees are just asked to point out their responsibilities and interaction with other colleagues. The security policy acquisition results in:

   - **Process acquisition**: represents a structural perspective containing activities, their sequence and data artifacts generated as result of activity and flowing as input to other activities.
   - **Role acquisition** shapes the functional part of the process, in other words associates actors with their activities.
   - **Security acquisition** imposes security restrictions on the actors in organization.

Having emphasized the importance of separation of process and security aspects, SPRINT also differentiates between structural and operational security aspect. Structural aspect of security can also be referred to as static, as it just determines the set of tasks and data objects and how they relate to each other in the process model. Structural aspect doesn't take into consideration any dynamic processes which take place in the system and actors/services performing this tasks and actions on data objects.

The operational aspect on the other hand defines constrains on tasks and data objects. Constraints may prescribe for example which actors (human being or service) are authorized to perform certain task or access data object.

As we can see from the Fig. 10, it integrates both, structural and operational security concepts. Structural ones are represented by concept of responsibilities, whereas operational impose restrictions onto these responsibilities.

[16] defines responsibility as a piece of data (or document artifact) to work with or a set of interconnected tasks which logically belong to a certain role. However there is no operational semantic (read, write, delete) concerned with these data or set of tasks. The main point here is to define concepts, which can be further assigned to roles and are subject to constraining. As an enterprise may represent a large hierarchical multilevel structure responsibilities (represented by a set of tasks or data objects) may be shared by multiple actors which in turn may be located on different levels of hierarchy. Furthermore, responsibilities are normally interconnected, thus producing responsibility bundles $\mathscr{R}$.

Constriction principle states that constraints may be defined for a group of tasks to keep an order of execution. The same is also true for creation of data objects as a result of group of tasks. Further the group of responsibilities with respective responsibility constraints may be assigned to one or multiple roles. There are several ways in order to promote reusability. The first one is to assign one responsibility

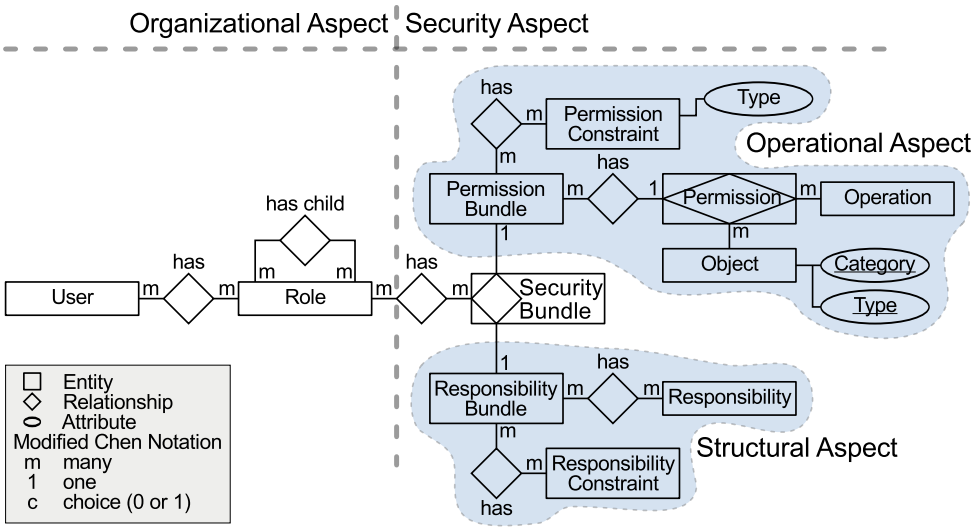


Figure 10: Security Policy Data Model by [16]

to multiple roles in the hierarchy and manage the appurtenant constraints for each role. That is not considered to be the most effective way. The other option for responsibility reuse is the inheritance of responsibilities by higher roles from their sub roles in a object-oriented manner with all the ensuing consequences.

Having made the structural security concepts clear, we get to more practical example. Any organization has a set of responsibility bundles $\mathscr{R} = b_1, ..., b_n$. Each responsibility bundle has a set of responsibilities $r = r_1, ..., r_n$ and a set of constraints $rc = rc_1, ..., rc_n$ associated with it. A responsibility may be associated with data object (rdata) or task (rtask) and restricted by responsibility constraint. Responsibility constraints may also be of two kinds: Responsibility Task Pattern Constraint rc^{tp} and Responsibility Relation Constraint rc^{r}. We will have close encounter with responsibilities later in chapter Use Cases 4.1.

As we have already mentioned, there should be a way to impose constraints on the order in which tasks are performed. That is achieved by rc^{tp}. There are several ways to represent the sequence of constraints. We take the one, proposed by [19], the Linear Temporal Logic (LTL) expressions [19].

As next, the relation between data responsibilities r^{data} and responsibility task pattern constraints rc^{tp} should be formulated. This newly formed constraint is called, as mentioned above, the responsibility relation constraint. There is also a possibility to bound the occurrence of a specific data responsibilities for a restricted set of task patterns.

3.2 Operational Aspect

The operational aspect has to do with permissions and their constraints as against responsibilities in the structural aspect. Separate permissions along with permission constraints are combined in a permission bundle $\mathscr{P}$. The goal of having permission constraints in the model is to impose restrictions onto responsibilities in responsibility bundles $\mathscr{R}$ from the structural part of the model.

In that way a permission bundle $\mathscr{P} = \{p, pc\}$ consists of a permission itself p and a set of permission constraints $pc = pc_1, ..., pc_n$. Permission points out which operations are allowed to be performed on security object. Permission is valid only for the whole responsibility bundle. Next, permissions constraints restrict the permission, which in our model describe a certain situation, where it should be checked, whether these or those permission constraints should be applied. As we already have discussed in introduction, there may be different situation in which we should check the various environment conditions:

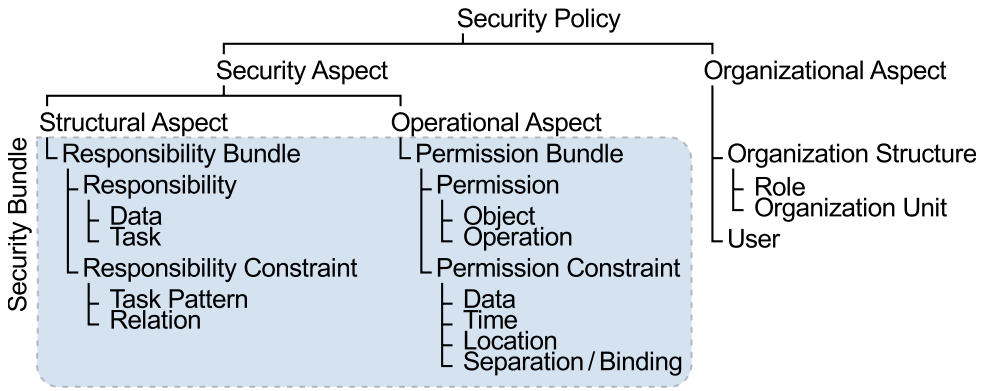


Figure 11: Security Policy - Overview and Definition by [16]

geographical location, time periods and so on. That's why we distinguish between data constraints, time constraints, location constraints and separation/binding constraints. Here are more precise definitions of each of them.

Data constraints pc^d are used to restrict a data responsibility r^{data} depending on the value of data.

Time constraints pc^t are used to restrict task responsibility r^{task} depending on intended execution time for that task.

Location constraints pc^l restrict task responsibilities rtask depending on geographical location of the actor, trying to execute this task or depending on location of the resource on which a task is intended to be performed.

Separations/binding constraints pc^{sb} specifies the need for assignment of different/same resources. This type or restriction is only applicable with the responsibility task pattern constraints rc^{tp}.

Depending on situation different kind of constraints may be activated. It is clear that in a situation, dealing with task execution no data constraints check is applicable.

Till now we have covered the structural and operations aspects, which together form the security aspect. Security aspect and organizational aspect in their turn comprise a **security policy** (Fig. 11).

This figure (11) lets us define a security bundle which is $\mathscr{S} = \{s_1, \ldots, s_n\}$ with $s_i = \{\mathscr{R}, \mathscr{P}\}$ defined as a single combination of responsibility and permission bundles. The relation between roles from organizational aspect and security aspects is defined as follows:

- each role can be related to 0...n security bundles.
- each security bundle can be associated with 1...* roles;
- different security bundles can combine one responsibility bundle with different constraints;
- security bundles can complement/override each other;

The concepts of responsibility and security bundles were introduced to reduce a complexity and implement the reuse functionality, as different approaches (like inheritance) may yield different problematic situations (too much power can be collected in one hands if using inheritance approach, deadly diamond of death and so on). Thus the one and the same responsibility bundle with different permission, associated with different actors from different levels in the hierarchy tree, may be used to control access of those actors to different resources. In the other case several responsibility bundles could be combined together to produce new desired virtual bundle without actual creation of the one.

3.3 Mapping of Responsibilities

Having discussed all the issues above we are now ready to see how the actual mapping of responsibilities and due to constraining process activities takes place. Normally this mapping is performed by the policy guardian, who should not necessary have a deep knowledge of responsibility or permission constraints. During the mapping responsibilities from the responsibility bundle should be associated with appropriate task (if responsibility is r^{task}) or data (if responsibility is r^{data}). At the same time a new mapping pair $m^{data}_{x,y}$ or which is defined by appropriate responsibility bundle id and responsibility id is created. The whole process lasts till all data and task objects are assigned to appropriate m^{data} or m^{task} respectively.

After all the responsibilities have been mapped, one might check if the process structure is consistent with the responsibility constraints. As these constraints don't describe the task order or task input/output, they just define a set of tasks (tasks may occur ad-hoc in this set) with associated data objects. This check is applicable in two cases: after finishing the mapping for a process or if a mapped process has undergone a change. This check may yield following errors:

- tasks are met in undefined order (no appropriate rc^{tp} for this order is found);
- known data objects are used or written with unspecified tasks (violation of rc^{r});
- known tasks are used with unknown data objects (violation of rc^{r});

The above errors may occur due to following reasons:

- the incorrect mapping was made by policy guardian;
- no additional mapping was undertaken after editing the process;
- Inconsistency of process with responsibilities and responsibility constraints.

The last reason may be solved by contacting the process designer with request to correct the process if such changes are needed. The process designer could have also unintentionally omitted the needed security policy. In this case this policy should be adopted and initial process redesigned to include the missing policy. As against checking for structural process security, which most definitely may be conducted automatically, the process correction and adoption of new security policies requires the policy guardian control.

3.4 Linear Temporal Logic

Since this master thesis often uses the Linear Temporal Logic expressions (LTL), we give a brief description of this approach. LTL is an acknowledged notation, which is used to define properties of reactive systems and is a subclass of Modal Logic [6]. It shows how an expression is described and evaluated over time. The result may be either true or false. In order to evaluate the expression different sets of operators (also called "modalities") are used. Among them are well known **logical or** ($\vee$) and **logical and** $\wedge$. In this thesis we use LTL expressions in order to compose constraints, which represent conditions necessary to perform certain action or access sensitive data. Moreover, we define our custom operators which are used in LTL expressions in order to reach a desired level of convinience while working with security aspects of certain processes.

4 Implementation/SPRINT-Editor

4.1 Use Cases

Since the bachelor degree of the author was about the computer supported control of nuclear power plants, in the first Use Case we are going to take a look at one of the core processes which take place in the nuclear power plant (NPP). As one might already know, modern NPPs represent very complex and spacious infrastructure with hundreds of workers per each reactor unit. The second business process will be dealing with bachelor and master enrollment processes. Let's start with the first example.

But before we begin to work with the target process, we would like to give a short generic description as to how to decompose process and outline main security aspects. There are different starting points for the process engineer to work with.

In some cases we don't have any initial data about enterprise and process. That means that we have to analyze process ourselves, find out core activities and other security sensitive tasks and data objects. In order to complete that step, strong domain knowledge is needed. That might also involve interviewing employees or workers who perform those tasks and work with artifacts. Crucial point here is not to miss any aspect that might help malicious actors to compromise the system. It is possible that tasks and data objects which have nothing to do with the company's core activities (at least directly) may be exploited by malicious actors. In other cases we might already have verbal or formal (e.g. Business Process Modelling Notation (BPMN)) description of the target process, with all important steps and data objects have been taken care of.

In the next four use cases we start with general description of processes. Here are the steps we take in order to get to formal representation of the security aspects of a process:

- Define roles in the process.
- Find tasks and data objects and prioritize them with regard to security concerns.
- Define structural dependencies between tasks (e.g. order), map data objects to task (data objects may either serve as input for the task or be generated as its output).
- Define which constraints are applicable for task or data object.
- Map groups of tasks and data objects and the attached constraints to roles.

Having performed all above steps we are ready to present the process in a formalized way which can be used for composing a BPMN diagram and further implementation using the web application we have developed.

And now back to our use cases. In order to provide some background knowledge about importance and role of nuclear energy in the modern world we are going to take a look at some facts:

- according to the data of International Atomic Energy Agency (IAEA) [1] today (10.07.2013) the number of power reactors in operations is as high as 434 and varies from 1 (Slovenia) to 100 (the United States);
- the total net installed capacity is 370 543 megawatts. To make it more understandable, one megawatt (MW) is enough to power up to 1000 homes. The major producers are the United States (98560 MW), France (63130 MW) and Japan (44215 MW);
- the total number of nuclear power reactors under construction is 68;

[1] http://www.iaea.org/pris/

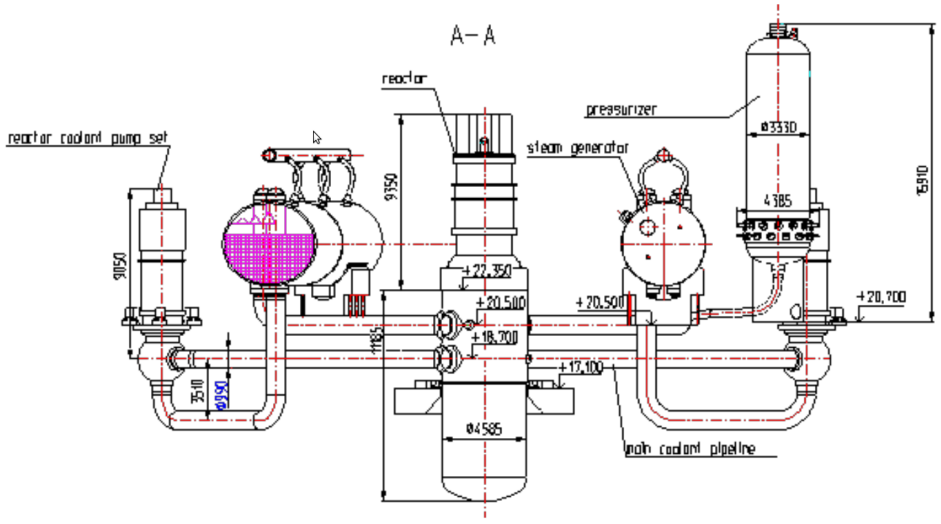


Figure 12: Layout of the Reactor Coolant System by

- the share of nuclear energy from the gross electricity production varies from just under 2% (1.99 in China) to 74.79% in France and plays one of the key roles in world leading production countries (see the list item above).

Nuclear energy has many advantages as well as drawbacks. The world knows quite a few atomic disasters, most recent and damaging being the Chernobyl and Fukushima nuclear accidents. At present time the world is seeking the way to substitute nuclear energy with safer and cleaner energy sources, but as of now nuclear energy remains indispensable.

As it was already mentioned, a modern NPP is a gigantic and very complex facility which requires permanent control and supervision. The simplified scheme of reactor with its coolant system may be seen on Fig. 12. Our first use case will consider the regular maintenance process which takes place every three years in the nuclear power unit (the time interval was taken for the VVER-1000 nuclear reactor – the most common type of reactors in the post-soviet nuclear industry). During this process the core of nuclear reactor, the turbine and steam generator undergo disassembling, cleaning and maintaining procedures.

The key part of the reactor unit (RU) maintenance stoppage is the replacement and relocation of the fuel elements which emit heat by means of fission reaction. The uranium assemblies burn out heterogeneously: the most of fuel is burned in the central part of nuclear reactor, whereas the outer assemblies contain more of unused uranium. The goal of service staff is to locate and replace fuel elements which are no longer suitable for heat generation, to relocate the array of outer and central fuel elements in order to reach the desired heat output.

Being potentially dangerous, there is also another participant in these processes, the International Atomic Energy Agency, which plays supervising and counseling function since 1957. As there are hundreds of people involved in this entire process, we are going to simplify it and define the core functions which are played by four major actors (Use Cases 1A).

4.1.1 Use Case 1A. Reactor Maintenance Stop

Having broken down the process into stages we get the following description: the VVER-1000 has to be shut down every 3 years in order to conduct the maintenance procedures. The procedure is initiated

by the Reactor Unit Chief Operational Officer, who issues the shutdown request and submits it to the Turbine Unit (TU) Chief Operational Officer (COO), the Nuclear Power Plant Chief Executive Officer and to the responsible IAEA supervisor. The request which is first approved by NPP CEO and then by the Turbine Unit COO is referred to as internal. The request sent to the IAEA is external one. Further actions are only possible after external and internal request have been approved by corresponding functionaries. Following deadlines should be met: internal request should be issued and approved not later than 30 days before the actual reactor unit shutdown is scheduled. IAEA request should be submitted 90 days before shutdown. The time passed since the last maintenance works should not exceed three years (provided that there were no emergency shutdowns and fuel element manipulations).

After settling the legal issues following routines are performed:

- fission reaction should be extinguished by means of injecting the graphite moderators in the reactor core. Coolant pump keeps on cooling reactor until the temperature drops from 316°Cto 60°C;
- turbine unit shutdown follows after that. Turbine is disconnected from the power grid; reactor lid is detached;
- fuel elements array is examined;
- fuel elements are relocated, the most depreciated fuel assemblies are removed and replaced with the new ones;
- the deprecated fuel assemblies are contained for at least 30 years;
- turbine, the coolant pump, steam generators and pipelines are cleaned;
- the system is reassembled after the maintenance works;
- the system is inspected and it's containment is checked;
- after having performed all checking procedures, startup request should be issued by the reactor unit COO, which should be again approved by the turbine unit COO, the NPP CEO and IAEA responsible functionary;
- nuclear reactor may be started again.

Once again, the whole procedure is extremely simplified due to enormous complexity and safety precautions and may not correspond exactly to the real world maintenance sequence.

In order to implement the above business process in our system, we should first define the in the next step the key tasks and data elements as well as point out the duties among the actors.

Actors:

- the reactor unit Chief Operational Officer (RU COO);
- the turbine unit Chief Operational Officer (TU COO);
- the nuclear power plant Chief Executive Officer (NPP CEO);
- the International Atomic Energy Agency responsible supervisor;

Data Objects:

- internal reactor unit stoppage request;

- internal reactor unit stoppage clearance/rejection;
- external (IAEA) reactor unit stoppage request;
- external (IAEA) reactor unit stoppage clearance/rejection;
- reactor stoppage log;
- maintenance log;
- fuel assemblies inspection and replacement report;
- internal start request;
- internal start clearance/rejection;
- external start request;
- external start clearance/rejection.

Constraints:

- $C1$: the NPP CEO, Reactor Unit COO, Turbine Unit COO and IAEA responsible supervisor are all different persons;
- $C2$: time since the last maintenance should not exceed 3 years;
- $C3$: internal stoppage request should be approved not later than 30 days before actual stoppage;
- $C4$: external stoppage request should be approved not later than 90 days before actual stoppage;
- $C5$: disassembling procedure may only begin after coolant liquid drops the temperature from 316°C to 60°C;

In the figure 13 we represent the process using the BPMN:

What we are going to do now is to map the steps from the above business model with the tasks elements which will go into the editor from implemented application. One important clarification should be made in this place: as it was already mentioned, we the task elements for editor aren't the exact representation of the tasks from the business model in BPMN notation; they rather emphasize actions from the point of view of security. We will also abbreviate some tasks from the BP model for the sake of compactness, as we put the task patterns in form of Linear Temporal Logic expressions. Next we put the task objects in the table 1.

As you can see, the formal representation of the tasks that goes into implemented editor differs from what we had in the BPMN model. Now we will map the data objects which will be added to the task patterns in the next step in the table 2:

The verbal description of the use case and the BPMN diagram are showing the roles present in our situation. But we are still missing users, i.e. actors, which are entitled with that roles. Table 3 puts all the roles, their brief description and real users appropriately.

Having defined all the crucial points in our first use case, we are ready to start composing constraints to implement the security aspects. As it was mentioned in the previous sections, the two security aspects that matter in the PAIS are the structural and the operational. The BPMN diagram in figure 13 depicts the structural devision of the target organization and also prescribes the order in which tasks are executed by the carrier of some certain responsibility (or role). What we are about to do now is to compose

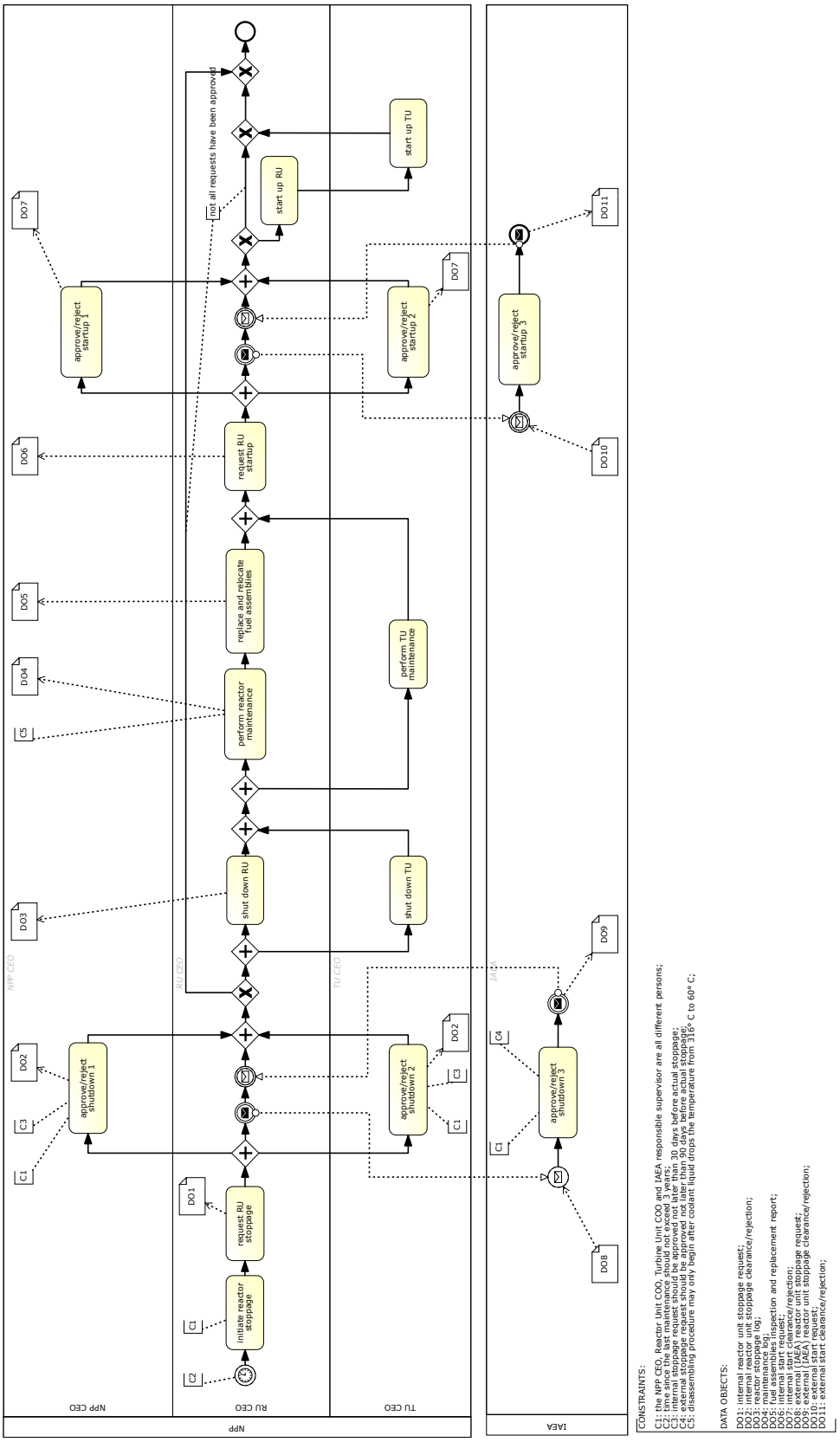


Figure 13: Use Case 1A: Reactor Maintenance Stop. BPMN Representation

Table 1: *Task Objects for Use Case 1A*

Task Nr.	Task from the BPMN Model	Task that goes into Editor
$T1$	initiate Reactor Stoppage	initStop
$T2$	request RU Stoppage (from the NPP CEO by means of submitting message to another 3 participants)	reqRUStop
$T3$	approve/Reject Shutdown 1 (NPP CEO lane)	app_rejSD1
$T4$	approve/Reject Shutdown 2 (TU COO lane)	app_rejSD2
$T5$	approve/Reject Shutdown 3 (IAEA lane)	app_rejSD3
$T6$	shut down RU	shutRU
$T7$	shut down TU	shutTU
$T8$	perform RU maintenance	maintainRU
$T9$	perform TU maintenance	maintainTU
$T10$	replace and Relocate Fuel Assemblies	fixFuel
$T11$	request RU start	reqStart
$T12$	approve/reject start 1	app_rejSt1
$T13$	approve/reject start 2	app_rejSt2
$T14$	approve/reject start 3	app_rejSt3
$T15$	start up RU	startRU
$T16$	start up TU	startTU

Table 2: *Data Objects for Use Case 1A*

DO Nr.	Data Object from the BPMN Model	Task that Goes into Editor
$DO1$	internal reactor unit stoppage request	intReq
$DO2$	internal reactor unit stoppage clearance	intClr
$DO3$	external (IAEA) reactor unit stoppage request	extReq
$DO4$	external (IAEA) reactor unit stoppage clearance	extClr
$DO5$	reactor stoppage log	stopLog
$DO6$	maintenance log	maintLog
$DO7$	fuel assemblies inspection and replacement report	fuelRep
$DO8$	internal start clearance	intStartClr
$DO9$	external start clearance	extStartClr
$DO10$	temperature of the liquid in the 1st reactor cooling circuit	cLiquid

the constraints which we will later input in our web application. Once again, the constraints will be represented in form of LTL expressions (see Section 3.4). It is important to mention that it is preferable to have a greater number of smaller LTL rules rather than one but long rule. This approach makes it easier to spot potential rule violation.

First of all we define the responsibility task pattern constraint, which prescribe which tasks are allowed to occur in the given process, the order in which task elements occur is irrelevant. As this pattern is going to be quite long, we brake it down into several smaller parts and then will combine them together in $b_{\text{reactor maintenance}}$, $rc^{tp}_{1.4}$.

Table 3: *Roles and Persons in Use Case 1A*

Role from BPMN Diagram	Brief Description	Users
NPP CEO	There is only one physical person entrusted with this role	Juergen Mangler
RU COO	At a given point of time there may be one or more reactor unit chief operational officers on the industrial site per reactor unit (depending on the complexity and internal structure). They normally work shift wise, but depending on situation and current needs may also cooperate to reach the best outcome	Alexander Kyshynevskyi Alexander Weber
TU COO	Same as for RU COO	Alexander Kyshynevskyi Stefanie Rinderle-Ma
IAEA	is an employee of external organization. There may be one or more people entrusted with this role	Tanja Sahaidak

$$rc^{tp}_{1.1} : initStop \wedge reqRUStop \wedge app\_rejSD1 \wedge app\_rejSD2 \wedge app\_rejSD3 \qquad (b_{\text{reactor maintenance}}, rc^{tp}_{1.1})$$

$$rc^{tp}_{1.2} : shutRU \wedge shutTU \wedge maintainRU \wedge fixFuel \wedge maintainTU \wedge reqStart \qquad (b_{\text{reactor maintenance}}, rc^{tp}_{1.2})$$

$$rc^{tp}_{1.3} : app\_rejSt1 \wedge app\_rejSt2 \wedge app\_rejSt3 \wedge startRU \wedge startTU \qquad (b_{\text{reactor maintenance}}, rc^{tp}_{1.3})$$

$$rc^{tp}_{1.4} : rc^{tp}_{1.1} \wedge rc^{tp}_{1.2} \wedge rc^{tp}_{1.3} \qquad (b_{\text{reactor maintenance}}, rc^{tp}_{1.4})$$

Of course we could have squeezed all the tasks in one rule instead of decomposing it into three parts and then gluing them together in $b_{\text{reactor maintenance}}, rc^{tp}_{1.4}$. The case is that to machine, which will parse those LTL expressions, the number of rules doesn't play any role. Furthermore, with larger number of small rules it is easier to spot a failure. However decomposing should also be made wisely without going to extremes.

The relation between data responsibilities (data objects) may be put as in $b_{\text{reactor maintenance}}, rc^{r}_{2.3}$. Due to considerable number of data objects we choose to break down the LTL rule into smaller parts just as we did previously for the task objects.

$$rc^{r}_{2.1} : intReq \wedge intClr \wedge extReq \wedge extClr \qquad (b_{\text{reactor maintenance}}, rc^{r}_{2.1})$$

$$rc^{r}_{2.2} : stopLog \wedge maintLog \wedge fuelep \wedge intStartClr \wedge extStartClr \qquad (b_{\text{reactor maintenance}}, rc^{r}_{2.2})$$

$$rc^r_{2.3} : rc^r_{2.1} \wedge rc^r_{2.2} \wedge rc^{tp}_{1.4} \qquad (b_{\text{reactor maintenance}}, rc^r_{2.3})$$

Please note that the last element in the rule $b_{\text{reactor maintenance}}, rc^r_{2.3}$ is actually the task pattern $b_{\text{reactor maintenance}}, rc^{tp}_{1.4}$. Constraint C_1 states that roles from different swim lanes from the BPMN process model should be assigned to different physical persons. In order to comply with our basic paper [16] we can refer to constraints as permissions. To put this in the LTL form we may take the first task from each swim lane and restrict their execution by means of separation of duty constraint $b_{\text{reactor maintenance}}$, permission 1:

$$pc^{sb}_{1.1} : (app\_rejSD1 \neq initStop) \wedge (initStop \neq app_rejSD2) \wedge (app\_rejSD2 \neq app\_rejSD3)$$
$$(b_{\text{reactor maintenance}}, \text{permission 1})$$

In the same time we also know that startup and shutdown are approved by the NPP CEO, who is one and the same physical person. That is why we complement the previous rule with the following one $b_{\text{reactor maintenance}}$, permission 2, which expresses the binding of duties:

$$pc^{sb}_{1.2} : (app\_rejSD1 = app\_rejSt1) \qquad (b_{\text{reactor maintenance}}, \text{permission 2})$$

As a nuclear power plant may have multiple reactor unit COOs per each power reactor and they work shift wise (or collaborate in some cases), it is important to note, that core maintenance procedures, which are the $shutRU$, $maintainRU$ and $fixFuel$ are performed by one and the same role carrier $b_{\text{reactor maintenance}}$, permission 3. For instance, if user Alexander Weber was performing the shutdown an maintenance of reactor unit, there is no way, that the other reactor unit COO Alexander Kyshynevskyi steps in in the middle of the process and relocates the fuel assemblies.

$$pc^{sb}_{1.3} : (shutRU = maintainRU) \wedge (maintainRU = fixFuel) \qquad (b_{\text{reactor maintenance}}, \text{permission 3})$$

The same is true for turbine unit COOs for the core maintenance tasks $shutTU$, $maintainTU$ $b_{\text{reactor maintenance}}$, permission 4.

$$pc^{sb}_{1.4} : (shutTU = maintainTU) \qquad (b_{\text{reactor maintenance}}, \text{permission 4})$$

Finally, the shutdown and startup approval/rejection on behalf of the IAEA responsible functionary should also be performed by one and the same physical person (e.g. Tanja Sahaidak) $b_{\text{reactor maintenance}}$, permission 5.

$$pc^{sb}_{1.5} : (app\_rejSD3 = app\_rejSt3) \qquad (b_{\text{reactor maintenance}}, \text{permission 5})$$

Constraint C_2, stating that time since last maintenance should not exceed three years, which means following ($b_{\text{reactor maintenance}}$, permission 1):

$$pc^t_2 : \text{curr}(fixFuel) - \text{prev}(fixFuel) \leq 3years \qquad (b_{\text{reactor maintenance}}, \text{permission 1})$$

This one requires a bit of explanation. Time constraint pc_2^t determines the time span, which has passed since last procedure of relocating and replacing the fuel assemblies. The function *curr* is representing the date when the current procedure is scheduled. *prev* function determines the time when the previous operation was performed.

Constrains C_3 and C_4 define the deadlines for reactor shutdown which should be made with regard to corresponding internal and external clearances ($b_{\text{reactor maintenance}}$, permission 1 and $b_{\text{reactor maintenance}}$, permission 1):

$$pc_3^t : (shutRU - app\_Rej1 \geq 30days) \wedge (shutRU - app\_Rej2 \leq 30days) \qquad (b_{\text{reactor maintenance}}, \text{permission 1})$$

$$pc_4^t : shutRU - app_Rej3 \geq 90days \qquad (b_{\text{reactor maintenance}}, \text{permission 1})$$

Our last constraint for this use case C_5 only allows the disassembling routine after the coolant temperature of 60 °C has been reached $b_{\text{reactor maintenance}}$, permission 1:

$$pc_5^d : cLiquid \leq 60^{\circ}\text{C} \qquad (b_{\text{reactor maintenance}}, \text{permission 1})$$

4.1.2 Use Case 1B. Reactor Contingency Stop

In the previous use case we went through the regular power reactor and turbine unit maintenance procedures, which normally involve four different parties. We also saw that in order to begin the whole routine, corresponding clearances should be issued by three of four participants. Furthermore different deadlines should be met for these clearances.

However, being an enormous and extremely complex facility, contingencies and unscheduled situations are inevitable while operation the NPP. In this kind of cases wasting time and requesting for shutdown approvals may lead to negative consequences. NPP internal policies provides special instructions for emergency reactor shutdown and maintenance. Let's see how they differ from regular routine:

- first of all the number of roles is shortened to two: Reactor Unit Chief Operational Officer and Turbine Unit Chief Operational Officer;
- there is no need to ask for permission from the NPP CEO and IAEA representative (may seem odd but if such situations happen at night, for example, the TU and RU COOs may be the only two of four people in previous example who are actually there). In that way the number of pools and lanes is shortened to one and two correspondingly;
- the RU COO just lets the TU COO know about the necessity to shut down the turbine unit (no permission from TU COO is needed here too);
- the number of tasks and data objects is also shrunk, we will see later how. Furthermore some new responsibilities will be added to the task list.

With that being said, we now go the same way as before and outline the key differences of this use case (Fig. 14) from use case 1A (Fig. 13).

The process diagram itself looks much more compact than the previous one. Instead of sixteen task elements we now have only eight, seven of which were taken from previous spacious model. The only

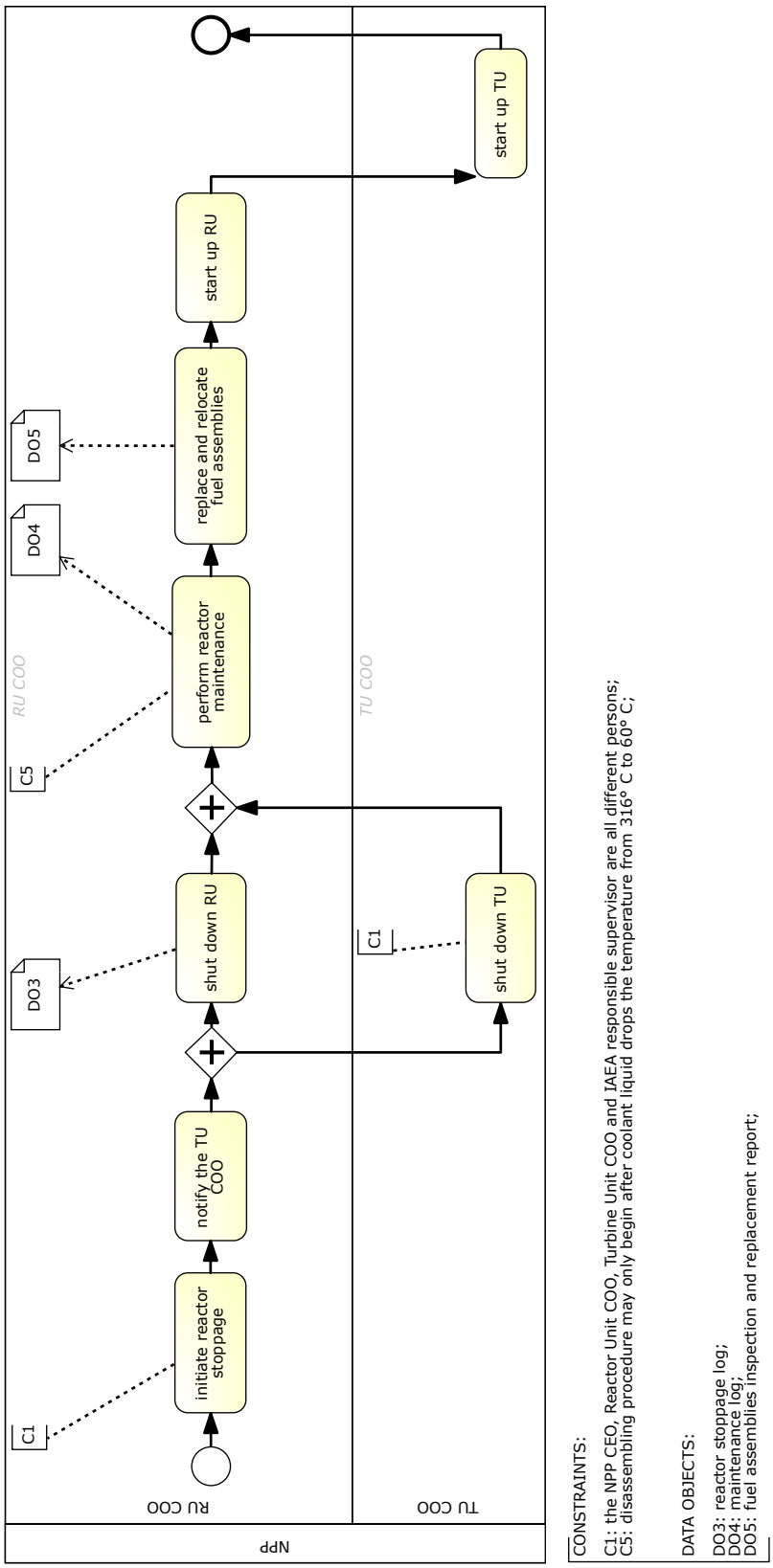


Figure 14: Use Case 1B: Reactor Contingency Stop. BPMN Representation

Table 4: *Task Objects for Use Case 1B*

Task Nr.	Task from the BPMN Model	Task that goes into Editor
$T1$	initiate Reactor Stoppage	initStop
$T6$	shut down RU	shutRU
$T7$	shut down TU	shutTU
$T8$	perform RU maintenance	maintainRU
$T10$	replace and Relocate Fuel Assemblies	fixFuel
$T15$	start up RU	startRU
$T16$	start up TU	startTU
$T17$	notify the TU COO	notifyTU

Table 5: *Data Objects for Use Case 1B*

DO Nr.	Data Object from the BPMN Model	Task that Goes into Editor
$DO5$	reactor stoppage log	stopLog
$DO6$	maintenance log	maintLog
$DO7$	fuel assemblies inspection and replacement report	fuelRep
$DO10$	temperature of the liquid in the 1st reactor cooling circuit	cLiquid

new task element in this case is there for the sake of communication between the two participants (table 4.

According to the table above, the task elements $T2$, $T3$, $T4$, $T5$, $T9$, $T11$, $T12$, $T13$ and $T14$ are excluded from the shortened business process. The same is true for data elements too. From the previous nine data objects we only have four: $DO5$, $DO6$, $DO7$ and $DO10$ (table 5).

As we already have almost all elements from the previous case (see Table 4), we can **REUSE THEM**. We only have to add the new task $T17$ *"notify the TU COO"* to the list on the backend. The responsibility pattern looks following way ($b_{\text{reactor contingency stop}}$, $rc^{tp}_{3.3}$):

$$rc^{tp}_{3.1} : initStop \wedge notifyTU \wedge shutRU \wedge shutTU \qquad (b_{\text{reactor contingency stop}}, rc^{tp}_{3.1})$$

$$rc^{tp}_{3.2} : maintainRU \wedge fixFuel \wedge reqStart \wedge startRU \wedge startTU \qquad (b_{\text{reactor contingency stop}}, rc^{tp}_{3.2})$$

$$rc^{tp}_{3.3} : rc^{tp}_{3.1} \wedge rc^{tp}_{3.2} \qquad (b_{\text{reactor contingency stop}}, rc^{tp}_{3.3})$$

The combination of data related responsibilities and previous pattern is shown in rule $b_{\text{reactor contingency stop}}$, rc^{r}_{4}:

$$rc^{r}_{4} : stopLog \wedge maintLog \wedge fuelRep \wedge cLiquid \wedge rc^{tp}_{3.3} \qquad (b_{\text{reactor contingency stop}}, rc^{r}_{4})$$

Again, the $rc^{tp}_{3.3}$ is the task pattern defined in previous step. In order to recap the task pattern just defines which tasks may occur during the process on the site; the rc^{r}_{4} shows which data objects are

needed or generated during that process. The sequence of the tasks is defined by the BPMN model in the figure 14

As of now we only have $C1$ and $C5$, both of which repeat the constraints from subsection above with the first one stating that actors in different swimlanes should be different physical persons and that disassembling of the reactor core should begin as soon as coolant liquid reaches the temperature of 60 °C. As the process structure differs slightly from what we had before, we will put it in appropriate form for the given use case.

$$pc^{sb}_{2.1} : shutTU \neq initStop \qquad (b_{\text{reactor contingency stop}}, \text{permission 1})$$

Next two rules express that reactor unit related tasks should be performed by one user (e.g. Alexander Weber):

$$pc^{sb}_{2.2} : initStop = notifyTU \qquad (b_{\text{reactor contingency stop}}, \text{permission 2})$$

$$pc^{sb}_{2.3} : (shutRU = maintainRU) \wedge (maintainRU = fixFuel) \qquad (b_{\text{reactor contingency stop}}, \text{permission 3})$$

The last rule expresses the temperature constraint

$$pc^{d}_{2.4} : cLiquid \leq 60°\text{C} \qquad (b_{\text{reactor contingency stop}}, \text{permission 4})$$

4.1.3 Use Case 2A. Bachelor Enrollment Process

For our next sample process we have chosen the enrollment procedure which is an example from academic area . The input data may be taken from the official university page[2]. This enrollment conditions and terms are taken for the winter semester 2013/14 and may vary for previous or upcoming winter semesters[3]. In general there are a few starting points for which applicants undergo different steps during the enrollment procedure. These are:

- the citizens of European Union (EU) and European Economic Area (EEA) member states;
- the citizens of other states not mentioned in previous list item.

Depending on applicant's citizenship, the list of documents may include some additional items, which require to be attested and translated. We also distinguish between bachelor and master studies applications. For master studies we also check whether an applicant already is an internal student at the University of Vienna. In the last case the procedure of applying for master studies takes a simplified form. Having discussed major points we now may plot decision tree diagram 15.

As one can see from the figure above the four arrows define four different situations which are possible during the enrollment phase. As our goal is to only show the appropriateness of the implemented

[2] https://studentpoint.univie.ac.at/vor-dem-studium/bachelor-bakkalaureatsstudien/?no_cache=1

[3] http://studentpoint.univie.ac.at/en/application/admission

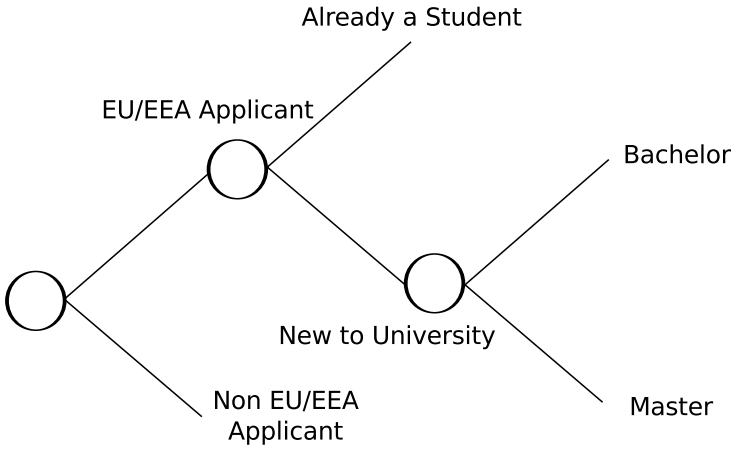


Figure 15: Use Case 2. Decision Diagram

web application for the purposes of handling security aspects in information systems, this particular use case doesn't claim to be complete and 100% precise. Therefore we omit some details, for example those situations with different (mostly prolongated) admission or Student Union fee deadlines for certain categories of applicants. With that being said we only concentrate our efforts on cases with EU/EEA applicants who may submit applications for either Bachelor or Master studies.

Let's now move on to the verbal process definition and description. According to StudentPoint[4] the generic process consists of following major steps:

1. potential applicant should pre-register oneself at https://erstanmeldung.univie.ac.at between the 1st of May and the 5th of September 2013;
2. after that the necessary list of documents should be gathered and prepared for the Admission Office in the University of Vienna;
3. a visit to University of Vienna admission office between the 17th of June and 5th of September is necessary to confirm enrollment. During this visit the documents from the step 2 should be handed in to the admission office clerk;
4. one should activate the student account at https://www.univie.ac.at/ZID/unet-aktivierung.
5. The final step which makes the Student Ticket valid is timely payment of the Student fee (in some cases the Student Union fee). The payment is due on September 5th.

Having performed the above steps, a student of the university may register oneself for preferred courses.

As we already mentioned, the procedure varies in case if applicant is already a student of the University of Vienna (or was a student some when in the past) and is willing to take another study. The difference here is that the pre-registration (step 1) can be skipped. During the first step some personal data should be entered. These are:

- login data:
  - the valid e-mail address (all university related e-mails will land here);

[4] http://studentpoint.univie.ac.at/en/application/admission

  - the desired password;

- study data:
  - the study program (bachelor, master, PhD, etc);
  - data about previous study;
  - desired study (e.g. Business Informatics);

- personal data:
  - personal details;
  - contact details;
  - statistical data;
  - as the result of performing pre-registration a student gets his/her matriculation number.

For the sake of compactness we omit the sub items of study and personal data and will just use "study data" and "personal data" data objects. According to Student Point[5] following documents should be handed in during the visit to the admission office:

- a valid passport;
- school leaving certificate;
- a certificate proving German proficiency level B2/2 or higher; (optional)
- social insurance number
- if applicable, an official document regarding change of name; (optional)
- a passport photography for the student ID.

All these documents will be treated as data objects during modeling and implementation phases. Here are the constraints applied to task and data elements. We will see those on the BPMN diagram as annotations:

- $C1$: an applicant and the admission office clerk are different natural persons;
- $C2$: pre-registration should take place between the 1st of May and the 5th of September 2013;
- $C3$: the personal visit to the admission office should take place between June 17th and the 5th of September 2013;
- $C4$: the student [union] fee should be paid after the submission of personal documents but before the 5th of September;
- $C5$: the passport photo should not be older than 2 years;
- $C6$: the German proficiency certificate should prove the level of at least B2;
- $C7$: login should be a valid e-mail address;
- $C8$: the password character sequence must contain between eight and sixteen characters and be a mixture of digits and letters.

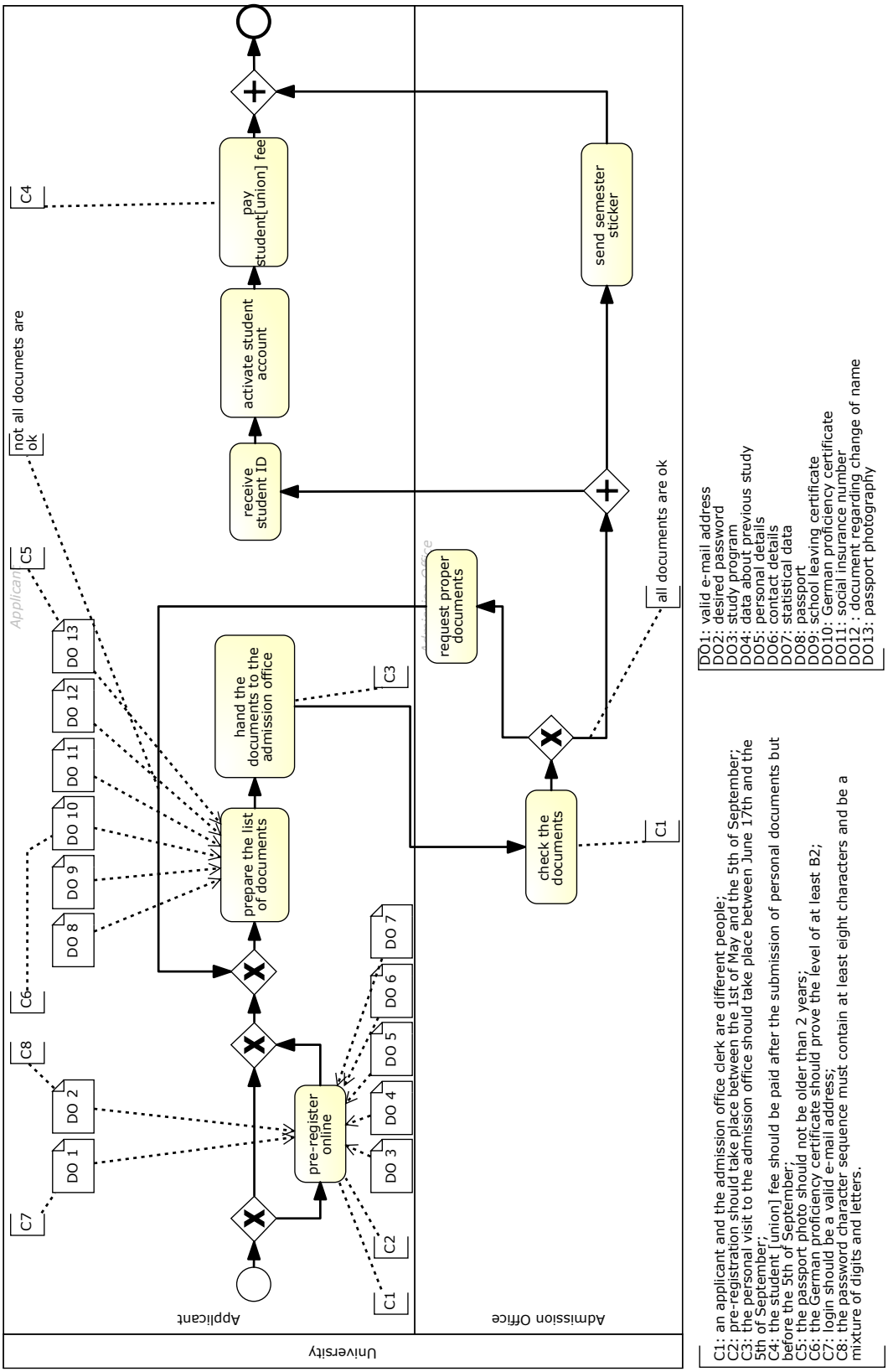


Figure 16: Use Case 2A: Bachelor Enrollment Process. BPMN Representation

Table 6: *Roles and Persons in Use Case 2A*

Role from BPMN Diagram	Brief Description	Users
Applicant	The person who is willing to study at the university	Alexander Kyshynevskyi Tanja Sahaidak
Admission Office Clerk	University employee, responsible for reception and checking the documents from applicants	Alexander Weber

Table 7: *Task Objects for Use Case 2A*

Task Nr.	Task from the BPMN Model	Task that goes into Editor
$T1$	pre-register online	preReg
$T2$	prepare the list of Documents	prepDocs
$T3$	hand in the Documents to the Admission Office	handInDocs
$T4$	activate student Account	enableAcct
$T5$	pay Student [Union] Fee	payFee
$T6$	check the Documents	checkDocs
$T7$	request Proper Documents	reqRightDocs
$T8$	issue student ID	issueID
$T9$	send Semester Sticker	sendSticker

The table 6 recaps the description of roles and adds the list of real users assigned to each role.

The final BPMN model may be seen in Fig. 16. Now let's map the artifacts from the BPMN diagram to the task and data elements in the actual application (Table 7):

Again we sacrifice the overall readability of the task elements in the application for the sake of compactness. Now we do the same procedure for the data elements in the table 8:

In order to recap the basics: the data and the task objects which were listed in the tables are referred to as permissions according to the paper [16]. The order in which tasks and data objects appear during the process execution is regulated by the BPMN diagram in Fig. 15, whereas the following rule deals with security aspects and only prescribes, which tasks are allowed to occur in the process $b_{\text{bachelor enrollment}}$, rc_1^{tp} :

$$rc_1^{tp} : preReg \wedge prepDocs \wedge handInDocs \wedge enableAcct \wedge payFee \wedge checkDocs \wedge issueID \wedge sendSticker \qquad (b_{\text{bachelor enrollment}}, rc_1^{tp})$$

The $b_{\text{bachelor enrollment}}$, rc_1^{tp} is showing the the use case where all necessary documents were submitted correctly and there was no need to request missing or incorrectly submitted papers. Next rule $b_{\text{bachelor enrollment}}$, rc_2^{tp} depicts situation where the admission office clerk is asking to provide correct or missing papers. Which consists of previously defined $b_{\text{bachelor enrollment}}$, rc_1^{tp} and another one task which is missing in the first rule.

$$rc_2^{tp} : rc_1^{tp} \wedge reqRightDocs \qquad (b_{\text{bachelor enrollment}}, rc_2^{tp})$$

[5] https://cs.univie.ac.at/home/news-events/sinfgleview/article/zulassung-zum-studium

Table 8: *Data Objects for Use Case 2A*

DO Nr.	Data Object from the BPMN Model	Task that Goes into Editor
$DO1$	valid e-mail address	eMail
$DO2$	desired password	passwd
$DO3$	study program	studyProgr
$DO4$	data about previous study	prevStud
$DO5$	personal details	persData
$DO6$	contact details	cntctData
$DO7$	statistical data	statData
$DO8$	passport	pass
$DO9$	school leaving certificate	schoolCert
$DO10$	German proficiency certificate	germanCert (optional)
$DO11$	social insurance number	socInsNum
$DO12$	document regarding change of name	nameChng (optional)
$DO13$	passport photography	passPhoto

Now we are able to reuse the both rules and present the general rule $b_{\text{bachelor enrollment}}$, rc_3^{tp}, which embodies both, the $b_{\text{bachelor enrollment}}$, rc_1^{tp} and the $b_{\text{bachelor enrollment}}$, rc_2^{tp} and is applicable for situation where necessary documents were submitted correctly and for those, requiring documents review.

$$rc_3^{tp} : rc_1^{tp} \vee rc_2^{tp} \qquad (b_{\text{bachelor enrollment}}, rc_3^{tp})$$

As we have quite a few data objects (or data responsibilities according to [16]), we can also split the big rule into several smaller ones and combine them afterwards. The relation constraint $b_{\text{bachelor enrollment}}$, rc_4^r is listing the data elements regarding pre-registration process:

$$rc_4^r : eMail \wedge passwd \wedge studyProgr \wedge prevStud \wedge persData \wedge cntctData \wedge statData \qquad (b_{\text{bachelor enrollment}}, rc_4^r)$$

The next rule ($b_{\text{bachelor enrollment}}$, rc_5^r) stands for the list of documents submitted during a visit to admission office:

$$rc_5^r : pass \wedge schoolCert \wedge germanCert \wedge socInsNum \wedge nameChng \wedge passPhoto \quad (b_{\text{bachelor enrollment}}, rc_5^r)$$

We have pointed out above, that some of the data objects are optional. These are contained in the rule $b_{\text{bachelor enrollment}}$, rc_5^r. The next one ($b_{\text{bachelor enrollment}}$, rc_6^r) covers cases where no optional documents are needed, i.e. we exclude some documents such as German proficiency proving document and name change certificate:

$$rc_6^r : rc_4^r \wedge pass \wedge schoolCert \wedge socInsNum \wedge passPhoto \qquad (b_{\text{bachelor enrollment}}, rc_6^r)$$

And finally we can cover both cases (with need for additional documents and without it) in constraint $b_{\text{bachelor enrollment}}$, rc_7^r:

$$rc_7^r : (rc_4^r \wedge rc_6^r) \vee (rc_4^r \wedge rc_5^r) \vee (rc_4^r \wedge rc_6^r \wedge germanCert) \vee (rc_4^r \wedge rc_6^r \wedge nameChng) \quad (b_{\text{bachelor enrollment}}, rc_7^r)$$

The first parentheses in rule $b_{\text{bachelor enrollment}}$, rc_7^r are handling case where no optional documents are needed, the second - where both optional documents should be submitted, the third - where only German proficiency certificate should be submitted and the last parentheses handling the case where name change certificate should be handed in. With that being said this final rule covers every possible situation regarding document lists. Now we combine constraints regarding task and data objects in pattern $b_{\text{bachelor enrollment}}$, rc_8^r:

$$rc_8^r : rc_3^r \wedge rc_7^{tp} \quad (b_{\text{bachelor enrollment}}, rc_8^r)$$

As you can see the last rule $b_{\text{bachelor enrollment}}$, rc_8^r looks quite simple, but includes all possible scenarios, covering situations with correctly and incorrectly submitted documents and necessities to submit optional documents. For simple use case we will make one last responsibility pattern assuming that all submitted documents are correct and there was no need to hand in optional ones (rule $b_{\text{bachelor enrollment}}$, rc_9^r):

$$rc_9^r : rc_1^r \wedge rc_6^{tp} \quad (b_{\text{bachelor enrollment}}, rc_9^r)$$

In this use case we have shown explicitly that one and the same rule may be put together in many different ways by means of reusing existing rules and combining together a few smaller rules. However it is important to avoid the overkill, where the number of rules explodes and becomes difficult to keep track of.

Now let us move on to permission constraints $C1$ - $C9$ which we have outlined for this particular use case. The $C1$, as in the cases before, states that participants in the swimlanes are different people. We won't deviate from the taken course and will express this constraint by means of separation of duty ($b_{\text{bachelor enrollment}}$, permission 1);

$$pc_1^{sb} : preReg \neq checkDocs \quad (b_{\text{bachelor enrollment}}, \text{permission 1})$$

Rules $b_{\text{bachelor enrollment}}$, permission 2 (Applicant lane) and $b_{\text{bachelor enrollment}}$, permission 3 (Admission office clerk lane are showing that task from one and the same swim lanes are performed by one and the same person (the optional task *reqRightDocs* was omitted):

$$pc_2^{sb} : (preReg = prepDocs) \wedge (prepDocs = handInDocs) \wedge (handInDocs = enableAcct) \wedge (enableAcct = payFee) \quad (b_{\text{bachelor enrollment}}, \text{permission 2})$$

$$pc_3^{sb} : (checkDocs = issueID) \wedge (issueID = sendSticker) \quad (b_{\text{bachelor enrollment}}, \text{permission 3})$$

$C2$ and $C4$ defines the pre-registration and fee deadlines (rules $b_{\text{bachelor enrollment}}$, permission 4 and $b_{\text{bachelor enrollment}}$, permission 5):

pc_4^p : control flow, *preReg*, execute
$pc_{4.1}^t$: 01.05 *till* 05.09
($b_{\text{bachelor enrollment}}$, permission 4)

pc_5^p : control flow, *payFee*, execute
$pc_{5.1}^t$: *before* 05.09
($b_{\text{bachelor enrollment}}$, permission 5)

$b_{\text{bachelor enrollment}}$, permission 6 shows $C5$, stating that passport photograph should'n be older than two years

pc_6^d : passPhoto not older than 2 yrs. ($b_{\text{bachelor enrollment}}$, permission 6)

The $C3$ regulates the time frame for documents submission to the admission office ($b_{\text{bachelor enrollment}}$, permission 7):

pc_7^p : control flow, *handInDocs*, execute
$pc_{7.1}^t$: 17.06 *till* 05.09
($b_{\text{bachelor enrollment}}$, permission 7)

Constraint $C7$ stands for checking the valid e-mail addresses. This may be done by simple free text or through regular expression. The last alternative is more reliable, however it is unknown which policy enforcement mechanisms will lie behind our application, so we present both variants in rules $b_{\text{bachelor enrollment}}$, permission 8 and $b_{\text{bachelor enrollment}}$, permission 9:

pc_8^d : *eMail* conforms to ($b_{\text{bachelor enrollment}}$, permission 8)

```
^\w+@[a-zA-Z_]+\.[a-zA-Z]{2,3}$
```

pc_9^d : *eMail* must be valid and contain '@' and '.' ($b_{\text{bachelor enrollment}}$, permission 9)

The last constraint $C8$ prescribes the length and allowed characters in the password $b_{\text{bachelor enrollment}}$, permission 10:

pc_{10}^d : *passwd* conforms to ($b_{\text{bachelor enrollment}}$, permission 10)

```
^[a-zA-Z]\w{7,15}$
```

This regular expression states, that the first character in a password must be a letter (upper or lower case), followed by at least seven and most fifteen characters, which may be either digits or letters or underscores. The next rule $b_{\text{bachelor enrollment}}$, permission 11 does the same in free text:

pc_{11}^d : *passwd* must begin with a letter, be between 8 and 16 characters long, and contain digits
($b_{\text{bachelor enrollment}}$, permission 11)

4.1.4 Use Case 2B. Master Enrollment Process

As in the use case with nuclear power plant, we are also going to have the second scenario here too. The goal of doing so is to show how previously defined elements may be reused. In this use case we will deal with master enrollment process, which is almost identical to bachelor enrollment except for couple of aspects.

First of, we describe the situation verbally. There are two possible starting points here: one with students with bachelor degree from Austrian universities and with Bachelor degree from non-Austrian universities. Moreover students with non European Union or European Economic Area Bachelor degree should meet some extra admission requirements.

In the first case students from Austrian university should just fill the application form and mail it to admission office. All other students should take following steps:

1. pre-registration online (same procedure as in Use Case 2A);
2. submit required documents to admission office;
3. in case of admission personally visit admission office to receive student ID. Student should present his (her) passport of another official ID, passport photograph and notification of admission;
4. pay tuition fee;
5. activate student account.

We have omitted the deadlines here. Even though the actual deadlines for master enrollment differ (enrollment period is extended up to 30th of November) implementation principles remain the same as in Use Case 2A. The personal data in pre-registration step is the same as in corresponding step of Use Case 2A. The list of documents from submitted in step 2 should contain following items:

- filled application form;
- previous bachelor diploma;
- transcripts for bachelor diploma;
- copy of passport;
- special certificate of special qualification for university studies (optional);
- certificate proving German proficiency level B2/2.

With regard to Use Case 2A we are able to reuse tasks $T1$, $T2$, $T3$, $T4$, $T5$, $T8$, $T9$. There are a couple of new ones coming into play, so the full list of tasks may be seen in Table 9.

The table of roles and persons remains the same (Table 6 from Use Case 2A), so there is no need to duplicate it here.

As for data objects we are able to reuse some objects from previous use case: $DO1$-$DO7$, $DO10$, $DO13$ (see Table 8). The complete list of data object for actual use case is represented in Table 10

The BPMN diagram of the process may be seen in Fig. 17.

As our goal here is just to show the reusal possibilities of the application, we will not blow out the rule description here one more time and will just present the final patterns.

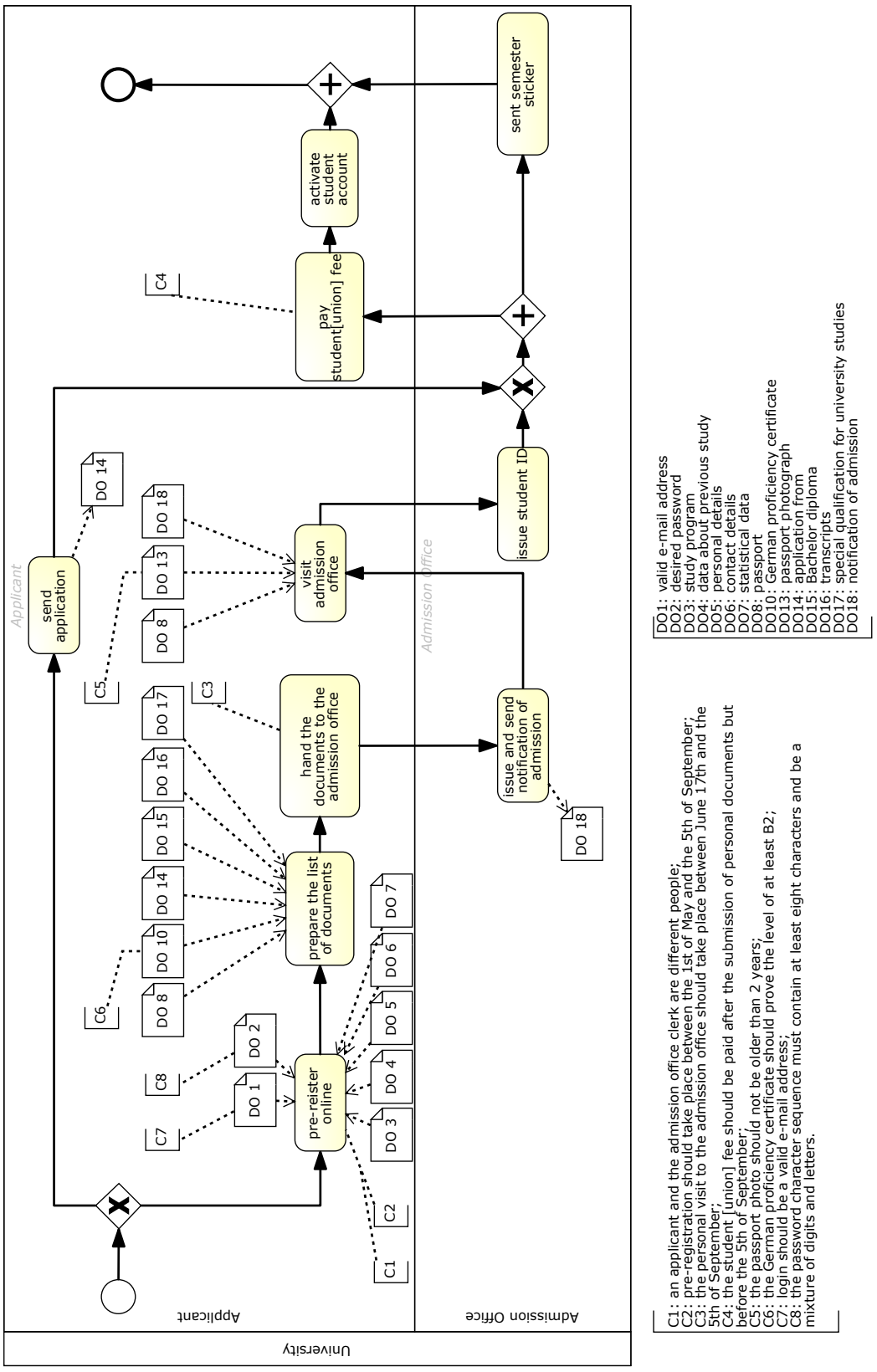


Figure 17: Use Case 2B: Master Enrollment Process. BPMN Representation

Table 9: *Task Objects for Use Case 2A*

Task Nr.	Task from the BPMN Model	Task that goes into Editor
$T1$	pre-register online	preReg
$T2$	prepare the list of Documents	prepDocs
$T3$	hand in the Documents to the Admission Office	handInDocs
$T4$	activate student Account	enableAcct
$T5$	pay Student [Union] Fee	payFee
$T8$	issue student ID	issueID
$T9$	send Semester Sticker	sendSticker
$T10$	send application	sendApp
$T11$	visit admission office	visitAO
$T12$	issue and send notification of admission	issueNoA

Table 10: *Data Objects for Use Case 2A*

DO Nr.	Data Object from the BPMN Model	Task that Goes into Editor
$DO1$	valid e-mail address	eMail
$DO2$	desired password	passwd
$DO3$	study program	studyProgr
$DO4$	data about previous study	prevStud
$DO5$	personal details	persData
$DO6$	contact details	cntctData
$DO7$	statistical data	statData
$DO8$	passport	pass
$DO10$	German proficiency certificate	germanCert (optional)
$DO13$	passport photography	passPhoto
$DO14$	application form	appForm
$DO15$	bachelor diploma	bachDipl
$DO16$	tanscripts	transcr
$DO17$	special qualifications for university studies	specQual
$DO18$	notification of admission	admNotif

$$rc_1^{tp} : preReg \wedge prepDocs \wedge handInDocs \wedge enableAcct \wedge payFee \wedge issueID \wedge sendSticker \wedge sendApp \wedge visitAO \wedge issueNoA$$
$$(b_{\text{master enrollment}}, rc_1^{tp})$$

The $b_{\text{master enrollment}}, rc_1^{tp}$ relation constraint is listing the tasks which must occur in the process. As for the data objects we have quite a lot of those to cram them into one relation constraint, so we split it into two smaller ($b_{\text{master enrollment}}, rc_{2.1}^{r}$ and $b_{\text{master enrollment}}, rc_{2.2}^{r}$) ones and after combine that combine in one final rule $b_{\text{master enrollment}}, rc_{2.3}^{r}$.

$$rc_{2.1}^{r} : eMail \wedge passwd \wedge studyProgr \wedge prevStud \wedge persData \wedge cntctData \wedge statData$$
$$(b_{\text{master enrollment}}, rc_{2.1}^{r})$$

$$rc^r_{2.2} : germanCert \wedge passPhoto \wedge appForm \wedge bachDipl \wedge transcr \wedge specQual \wedge admNotif \qquad (b_{\text{master enrollment}}, rc^r_{2.2})$$

$$rc^r_{2.3} : rc^r_{2.1} \wedge rc^r_{2.2} \qquad (b_{\text{master enrollment}}, rc^r_{2.3})$$

All constraints from use case 2A are valid here, so there is no need to put them here again. We will only adjust separation constraint $C1$ slightly as it depends on the newly introduces data objects ($b_{\text{master enrollment}}$, permission 1, $b_{\text{master enrollment}}$, permission 2 and $b_{\text{master enrollment}}$, permission 3).

$$pc^{sb}_1 : issueNoA \neq preReg \qquad (b_{\text{master enrollment}}, \text{permission 1})$$

This constraint shows that actors from different swimlanes are different natural persons.

$$pc^{sb}_2 : (preReg = prepDocs) \wedge (prepDocs = handInDocs) \wedge (handInDocs = visitAO) \wedge (visitAO = payFee) \wedge (payFee = \qquad (b_{\text{master enrollment}}, \text{permission 2})$$

$b_{\text{master enrollment}}$, permission 2 reflects the binding constraint for the applicant swimlane, whereas $b_{\text{master enrollment}}$, permission 3 does the same for admission office clerk swimlane.

$$pc^{sb}_3 : (issueNoA = issueID) \wedge (issueID = sendSticker) \qquad (b_{\text{master enrollment}}, \text{permission 3})$$

4.2 User Interface

The target application is implemented as web application and follows its main principles: the client is accessing the functionality through a web browser by means of reading/saving rules from/on the backend server. We have chosen this architecture in order to make the handling more easy and comfortable for end user, who in our case just needs the web browser in order to begin working with the target program. So far the functionality of the program is optimized for the gecko based web browsers, the most popular of which is the Firefox. In order to reach the best results it is advised to update the Firefox to the newest version, which is 23 for the Linux environment as of today (23.10.2013) [6]. Because of the architecture we have chosen, there is no need to install any development kits or other programs for the end user: the functionality can be accessed from any machine, connected to the Internet. The same is also true for the implementation phase we were going through: by means of git, the version and content management software, the source code can be easily accessed, modified and maintained from anywhere.

As the main programming language the JavaScript (JS) was chosen. This choice was made because all modern mainstream web browsers support JavaScript. Next idea behind this choice is that JS and its library JQuery are perfectly suited to support operation with the Document Object Model (also known as DOM) of the HTML document, which allows for quick search of elements of specific type, set their representation attributes, delete, clone, insert new ones, etc. Moreover, with JS the processing of the DOM takes place on the client machine. That is much quicker than server sided processing, requiring sending HTTP request and receiving responses each time, when the DOM needs to be changed. And finally JS is event-based programming language, which provides a wide range of possibilities to work

[6] https://www.mozilla.org/en-US/firefox/central/

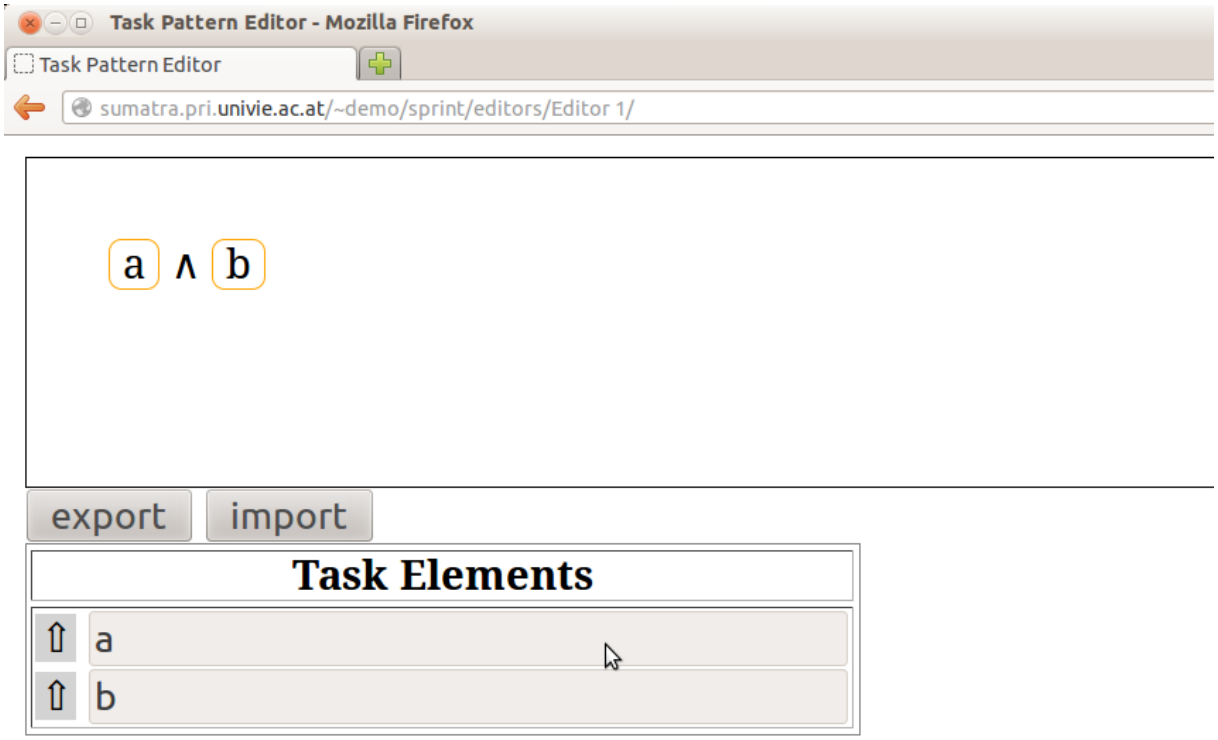


Figure 18: Initial look of the editor

with events, generated by user while clicking, rolling over, dragging, calling the context menu, etc. on one of the DOM elements. With JS we are able to fine tune the behavior for each element.

According to exposee three editors should be implemented, whose goal is to guide and to support users in managing the security aspects:

- the **Task Pattern Editor** will describe which tasks may be used for certain process;
- the **Responsibility Editor** the data elements will be defined which are allowed to be used or are generated during the process execution (the process definition will be taken from Task Pattern Editor);
- the goal of the **Constraints Editor** is to impose constraints for data and task elements and for patterns which are the composition of data or task elements (both data and task elements are taken from previous two editors).

It should be noted that all these components are reusing one common part, which is the rule editor (see Fig. 19), which is instantiated in each of the above editors and customized in order to meet the requirements of each particular case. Furthermore, the implementation of the above editors follows principles described in [16]. That means that the editors are there to support certain steps from the paper, which have to do with managing security policies. In the use case sections above we have given the detailed description of how exactly security policies are defined all the way from verbal description of the business process, through the BPMN model down to the linear temporal logic rules, which play one of the core roles in achieving our goal.

We have already mentioned the goal is to implement the three editors which guide the user through the process of management of security policies. All three editors are using one common part, namely the

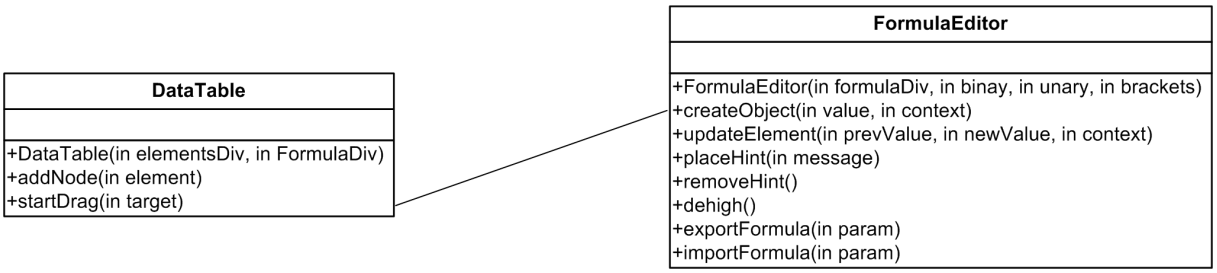


Figure 19: Class Diagram and Interaction between Main Classes

the rule editor (can be seen in the Fig. 18), which is slightly modified in order to comply with the needs from each step.

As the result we gave detailed description of the programmatic part, responsible for representation of the organizational structure with roles and users. Being parts of one superior SPRINT project, our target application and application, described above may be merged in the future works to operate in the bundle, thus contributing to creation of one robust system.

In order to recap the material and make it more clear let's go through all relevant security aspects once again:

Relevant Security Aspects

- organization - is an entire body whose security aspects we are taking care of;
- organizational unit - a part of organization, defined by its functions (e.g. accounting, marketing, research and development, etc.);
- responsibilities - abstract concept, which includes certain privileges and rights to perform some tasks or to work with artifacts (e.g. documents, equipment, hardware, etc.);
- role - another abstract concept which is entrusted with some set of responsibilities;
- user - a person (in some cases can also be a software unit, like web service) who obtains responsibilities by means of roles. One user may have from one to many roles, as well as role may be carried by more than one user.

As it was specified above, we have deconstructed the process of managing security policies into three steps, each of which has corresponding sub application. The core of each sub application is the rule editor, which in every case is fine tuned to meet specific requirements and is comprised by the JavaScript class [7]. Next important part is the table of elements, which are used to compose the rule. The table of elements is also represented by the JS class, which is responsible for loading the list of elements from backend server and converting them from XML to HTML format. The interaction of these two main classes may be seen at Fig. 19. The connecting line between two classes is not there to show the relation or multiplicity in terms of UML. Instead it just points out, that as soon as user has chosen the element from the table and has started to drag it onto editor area, the object which goes into formula is created by the FormulaEditor class. This object pops up on the screen as new formula operand when user releases the mouse button. Here is a brief description of the functions in `DataTable` class.

[7] The source code may be found at http://sumatra.pri.univie.ac.at/~demo/sprint/

- `DataTable(elementsDiv, formulaDiv)` - is the class constructor, responsible for loading the corresponding set of elements from the backend server and representing them in a tabular form. The two parameters are representing the names of the `<div>` elements where the elements table go.
- `addNode(element)` step by step adds elements to the DOM structure of the HTML document. The parameter is a String value which will display the name of the element in the list.
- `startDrag(target)` - triggers visualization process as user starts to drag element from the table in order to insert it into the formula above.

Here is what `FormulaEditor` class functions do:

- `FormulaEditor(formulaDiv, binary, unary, brackets)` - class constructor, which sets up the part of the HTML document representing the LTL rule. Parameters are representing the name of the `<div>` element which will hold the editor area, the last three are standing for sets of binary and unary operators as well as for possible brackets.
- `createObject(value, context)` creates new object to insert into the formula as user starts to drag the element from the table in the lower portion of the HTML document.
- `updateElement(prevValue, newValue, context)` - if user changes the name of the element in the table of elements, this function updates the name of the element in the formula above.
- `placeHint(message)` - may be tuned to display pop-up hints when user hovers over some specific type of elements in the formula.
- `removeHint()` - removes pop-up hints if there are any.
- `dehigh()` - removes highlighting from elements in the formula.
- `exportFormula(param)` - is responsible for converting the formula into XML and exporting it onto backend server.
- `importFormula(param)` - is used in the second editor in order to import previously composed task patterns into the editor. The patterns are converted back from XML into HTML.

As we have already mentioned, application is based on JS event model and supports large number of different events. The most relevant ones are listed in the table 11. For the sake of providing better understanding of the event model, we briefly go through the basics. All elements on the web page are objects of hierarchical document object model (DOM), with the topmost element being **Document**. Events in JS are following from uppermost hierarchy object to the target one. Having reached the target, event is propagated back all the way up to Document. By means of JS we may tune each element to react specifically on certain type of events. Thus we suppress the calling of standard context menu on the web page, when user is performing right clicks on data/task objects, substituting it for context specific user-defined menu. There are two important aspects to take into consideration while working with JS events, namely the event capturing and event bubbling. The event capturing is the method to intercept event as it is traveling through the levels of hierarchy to the target object. Event bubbling stands for capturing event after it has reached the target object and is on its way to the uppermost object, the Document. Understanding and utilizing these principles plays key role in our application.

In order to create the new editor, four documents are needed:

Table 11: *Events Supported by Editor*

Element	Supported Events
`document`	`ready` - event listener, which initializes necessary variables classes right after document is loaded in a browser
`"#export"`	`click` handles mouse click on the button. Exports the document onto backend server
`".draghandle"`	`mouseDown` is a starting point of dragging a new element onto the formula portion of the HTML document
`"span.binary"`	`click` toggles binary operator
`".dataElement", ".taskElement"`	`contextmenu` suppresses the browser default context menu and displays dynamic user-defined context menu
`".brackets"`	`contextmenu` displays context menu for brackets `mouseenter` highlights the matching bracket for the one being hovered over
`".unary"`	`click` toggles the unary operator

- *main.js* - this document is responsible for initialization of the two JS classes, which are contained in the next two files. Important point are the parameters, which are passed to the constructors of the classes, as they define the behavior of the future class instances.
- *formula_editor.js* - is file with editor class definition. This part is responsible for composing the LTL expression and manipulations with elements, which are already in the editor portion of the HTML document;
- *data_table.js* - is the file, responsible for loading the list of objects from the backend server and presenting them in necessary form. The loaded objects serve as building blocks for LTL rules, composed in the editor area of the document;
- *formula_editor_parser.js* - is the file, containing small function, which parses the LTL rule after it has been manipulated. The main goal of the function in this file is to spot and delete empty binary operands, thus making the LTL expression well formed;
- *index.html* - is the HTML document, which embeds the functionality of the JS classes. Besides the upper *.js files index.html should also include the script tag for loading the JQuery library. index.html is presented in the Listing 1;

```
1  <!doctype html>
2  <html>
3    <head>
4      <title>Task Pattern Editor</title>
5      <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.5/jquery.min.js"></
           script>
6      <script type="text/javascript" src="JS/formula_editor.js"></script>
7      <script type="text/javascript" src="JS/data_table.js"></script>
8      <script type="text/javascript" src="JS/formula_editor_parser.js"></script>
9      <script type="text/javascript" src="JS/main.js"></script>
10   </head>
11
12   <body>
```

```
13      <div id="formel"></div>
14      <div>
15        <button id='export'>export</button>
16      </div>
17      <div id="elements">
18        <table id="elementsTable" border="1">
19          <thead>
20            <tr>
21              <th>Task Elements</th>
22            </tr>
23          </thead>
24          <tbody>
25            <tr>
26              <!--Task Elements-->
27              <td id="taskList">
28                <ul class="elementsList">
29                  <li>
30                    <span class="draghandle" title="click here to drag">&#x21e7;</
                          span>
31                    <input class="taskName" placeholder="input name here ..."/>
32                  </li>
33                </ul>
34              </td>
35            </tr>
36          </tbody>
37        </table>
38      </div>
39    </body>
40  </html>
```

Listing 1: The index.html File

Regarding the markup of the index.html file, there should be two `<div>` areas where the editor and the list of elements will be loaded. Depending on the situation and needs the elements `<div>` should be marked up correspondingly. Thus, for example, in the first editor in our web application only the list of task elements should be loaded, which is arranged as HTML table with only one column. In the second case we have to load the list of data objects and the list of task patterns. Here the table with two columns has to be set up. The names of these two `<div>`-tags are going to be passed as arguments to JS class constructors along with some other arguments which we are about to look at in Listing 2.

```
1   var formulaArea = '#formel';
2   var listArea = '#elementsList';
3
4   var binary = {
5     'and': '∧',
6     'or': '∨'
7   };
8   var unary = {
9     'diamond': '◇',
10    'box': '□'
11  };
12
13  var brackets = {
14    'round': [' ( ',' ) '],
15    'tag': [' < ',' ] ']
16  };
17
18  myeditor = new FormulaEditor(formulaArea, binary, unary, brackets);
19  mytable = new DataTable(listArea, formulaArea);
```

Listing 2: Instantiation of the Editor and Table Classes

This piece of code is taken from the file *main.js*. This file handles editor adjustments, creation of class instances and assigning them to class reference variables. Lines 1 and 2 initialize two variables with string values, representing the HTML `<div>` tag IDs for the LTL rule area and the elements list area correspondingly. The lines 4 - 16 are defining the operators which the future editor will support for its elements. Thus 2 shows the code snippet from the first editor, which is meant to support the two logical operators and two binary ones. Furthermore, in order to define the operation precedence, the set of brackets is supported too. Here we omit the semantical meaning behind all those operators and brackets, as our goal now is to only explain how the editor should be set up.

After having covered the basic aspects of the editor functionality, it is clear, that this approach allows for very flexible and fast creation and tunning of new editors with needed specification. Thus, for example, the list of operators and brackets may be changed in a blink of an eye. If there is no need for some type of operators (e.g. no unaries are needed), the new class instance may be created by calling constructor with unary parameter set to **null**. We followed this way in the constraint editor (Editor 3), where we had no unary operators and only one binary, which was the comma for enumeration.

In order to recap the material, let's repeat the steps, which lead to creation of the editor once again:

1. create the HTML document (e.g index.html);
2. embed the JavaScript files *main.js, formula_editor.js, data_table.js, formula_editor_parser.js* into the HTML `<head>` portion of document ¡script¿ tag;
3. embed the JQuery library into `<head>` of the HTML. We recommend to use JQuery v1.5, as the functions in different versions of JQuery may behave differently;
4. create two `<div>` elements in the HTML document for editor and element areas and give them unique IDs;
5. depending on needs mark up the list `<div>` in the web document so that it contains the necessary number of columns. The code inside column can be seen in the Listing 1;
6. set up the *main.js* file to support certain set of binary, unary and bracket operators.

In order to make editor function properly some adjustments should be made in the *formula_editor.js* file as well. This file uses the names of the columns in the elements `<div>`, which carries semantic meaning, as we distinguish between the types of elements that we load from the backend server. In the Table 11 we present the exact description of functions and events supported by the web application.

Another one feature which was implemented but is not used as of now, is the ability to select multiple elements. That may be achieved by means of holding the `ctrl` key and clicking first and then last element of the group. As a result, elements will be highlighted with their border glowing light green.

We would also like to say couple words about the format of LTL rules. In order to present them in the browser window we use `<span>` elements, which carry no semantical meaning for the page markup and are used only for the purpose of formatting. These `<span>` elements get different class names and are are ordered hierarchically. Such format makes it easier to convert HTML object to XML. The following two listings are comparing the HTML and converted XML format (Listings 3 and 4).

It should be noted, that the top level node's name hard coded for each editor and carries semantic role. In that way the top level node in the Listing 4 is named *"responsibility"*. The further conversion happens as follows: function parses the HTML code recursively, paying attention to the number of classes. If HTML node only has one class name attached, that class name becomes the name of XML node. If further class name is provided, it becomes the value of the *type* attribute. So far the HTML nodes in the code represent maximum two classes. The further introduction of class names will require the revision of the conversion function.

```
<div id="formel">
  <span class="binary and">
    <span class="binaryOperand">
      <span class="dataElement" data-content="extReq" data-url="undefined">extReq</span>
    </span>
    <span class="binaryOperand">
      <span class="binary and">
        <span class="binaryOperand">
          <span class="dataElement" data-content="intClearance" data-url="undefined">intClearance</span>
        </span>
        <span class="binaryOperand">
          <span class="dataElement" data-content="intReq" data-url="undefined">intReq</span>
        </span>
      </span>
    </span>
  </span>
</div>
```

Listing 3: LTL rule in HTML format

```
<responsibility>
  <binary type="and">
    <binaryoperand>
      <dataelement>a</dataelement>
    </binaryoperand>
    <binaryoperand>
      <binary type="and">
        <binaryoperand>
          <dataelement>b</dataelement>
        </binaryoperand>
        <binaryoperand>
          <dataelement>c</dataelement>
```

```
13          </binaryoperand>
14        </binary>
15      </binaryoperand>
16    </binary>
17  </responsibility>
```

Listing 4: LTL rule in XML format

4.3 Backend

At the beginning of the Implementation section we mentioned that our application relies on a client-server architecture. In our case we decided that client and server communicate via REST. In order to provide some understanding in this field, we give a short description of REST. REST follows the principles of World Wide Web where any existing resource may be accesses by means of unique ID - the Uniform Resource Identifier (URI). The key principle of REST is also a *resource* [11]. The goal of this architectural approach is to simplify access to those resources. It is important to note that REST is not bound to some specific programming language, platform or even protocol (though most actively used one is HTTP). Instead that is just a set of prescriptions of how resources should be accessed and manipulated. Each programming language implements the principles of REST in its own way. Further important point is the *statelessness* of REST. That means, every request should also carry full information about the object resource. To put it other way server is not aware of any prior requests from the client. For example client is requesting the list of customers from the server in first request and then wants to obtain and modify information regarding most active customer in further requests. In order to do that second and further client requests should also carry information (ID, URI, etc) about the customer whose attributes are being accessed and modified. There are four main types of requests in REST:

- GET - is used to obtain the representation of specific resource;
- PUT - replaces the existing representation of resource or creates new resource;
- POST - creates new resource with given representation;
- DELETE - deletes resource.

Having explained briefly the principles of REST we give the description of the backend server for our application. All necessary data data is hosted on the university server[8]. It is loaded by the JS application in every editor depending on current needs. The root directory of server is shown in Fig. 20. The subdirectories contain XML files with editor-related data. Here is the brief description of the subdirectories content:

- `tasks` directory contains the list of task elements, which are used in the first editor for composing task patterns. The task patterns are also stored in this directory and are available under `http://sumatra.pri.univie.ac.at/~demo/sprint/backend/tasks/patterns`;
- `data-elements` stores the the list of data elements added to the task patterns, composed using first editor;
- `bundles` will hold xml files with constraints, defined in third editor. The structure of this xml is somewhat more complex than in previous two cases;

[8] `http://sumatra.pri.univie.ac.at/~demo/sprint/backend/`

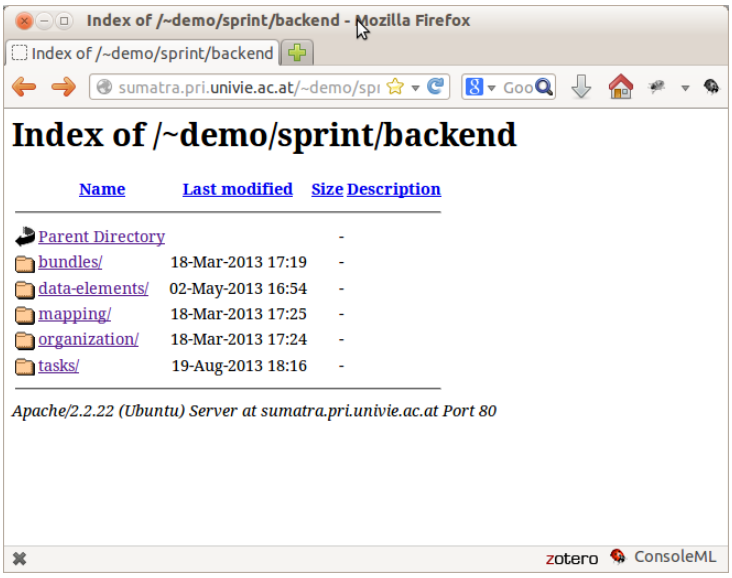


Figure 20: Root Directory of the Backend Server

Table 12: *Request Supported by the Server*

Resource	Supported Requests
`/tasks`	GET
`/tasks/{id}`	GET
`/data-elements`	GET
`/data-elements/{id}`	GET
`/tasks/patterns`	POST
`/bundles/permission/`	POST

- `organization` subdirectory will contain data about structure organization, its subdivisions, roles and users;
- `mapping` will represent data about mapping of roles to organizational units, roles to users and users to organizational units as was seen from the paper [12]. That information will be completed by mapping of constraints to roles (responsibilities).

The table 12 shows which types of REST requests are supported by the backend

In order to load the necessary data into browser window, the dedicated JS function retrieves information from server by means of sending GET-requests to server. The data from backend server is processed and converted to appropriate format. As we have already mentioned, the implementation of REST varies from one programming language to another. In the code snippet in Listing 5 we use the function of the JavaScript JQuery library. There is also JS native way to send requests, but it is a tiny bit more complex. It is important to notice that during the POST request we just send the pattern which should be added to the list of existing patterns. The rest is being handled by the server-side PHP code, which calculates the number of existing patterns and generates appropriate ID for the newly sent pattern.

```
1    $('#export').click(function(){
2      var exp = myeditor.exportFormula($(formulaArea));
```

```
    exportedF = exp[0].innerHTML;
    $.ajax({
      url: 'http://sumatra.wst.univie.ac.at/~demo/sprint/backend/tasks/patterns',
      type: 'POST',
      data: {
        xml: exportedF
      }
    });
  });
```

Listing 5: POST-Request from JS code

For security and test purposes we chose to update existing patterns or deleting them manually on the server. With that being said, we move on to data format representation for each concrete editor. As editor-related data is stored in form of XML, we chose the RELAX NG schema language to describe the data structure. The following two listings (Listing 6 and 7) show the structure for the first editor.

```
<?xml version="1.0" encoding="UTF-8"?>
<tasks>
  <task endpoint="a">
    <label>aliasA</label>
    <label>aa</label>
  </task>
  <task endpoint="b">
    <label>aliasB</label>
    <label>bb</label>
  </task>
</tasks>
```

Listing 6: Tasks XML representation

```
<element name="tasks" xmlns="http://relaxng.org/ns/structure/1.0">
  <zeroOrMore>
    <element name="task">
      <attribute name="endpoint">
        <text/>
      </attribute>
      <oneOrMore>
        <element name="label">
          <text/>
        </element>
      </oneOrMore>
    </element>
  </zeroOrMore>
</element>
```

Listing 7: Tasks RELAX NG Schema

As one might see, each task in the list has an attribute *"endpoint"*, which is unique and from one to many labels. As the list of tasks is loaded in the first editor, each task is represented by its first label.

The task pattern XML example and its RELAX NG schema, representing simple pattern $tp_{\text{Simple Task Pattern}}$, rc^{tp} showed in the next two listings (Listing 8 and 9)

$$rc^{tp} : a \wedge (\diamond b) \qquad (tp_{\text{Simple Task Pattern}}, rc^{tp})$$

```
<?xml version="1.0" encoding="UTF-8"?>
<patterns>
```

```
  <pattern id="0">
    <binary type="and">
      <binaryoperand>
        <dataelement>initiateStop</dataelement>
      </binaryoperand>
      <binaryoperand>
        <brackets type="round">
          <unary type="diamond">
            <dataelement>reqRUStop</dataelement>
          </unary>
        </brackets>
      </binaryoperand>
    </binary>
  </pattern>
</patterns>
```

Listing 8: Tasks Pattern XML representation

```
<element name="patterns" xmlns="http://relaxng.org/ns/structure/1.0">
  <choice>
    <element name="taskelement">
      <text/>
    </element>
    <element name="unary">
      <attribute name="type">
        <ref name="taskelement"/>
      </attribute>
    </element>
    <element name="brackets">
      <attribute name="type">
        <text/>
      </attribute>
      <choice>
        <ref name="taskelement"/>
        <ref name="unary"/>
        <ref name="brackets"/>
        <element name="binary">
          <attribute name="type">
            <text/>
          </attribute>
          <element name="binaryoperand">
            <choice>
              <ref name="taskelement"/>
              <ref name="unary"/>
              <ref name="brackets"/>
              <ref name="binary"/>
            </choice>
          </element>
          <ref name="binaryoperand"/>
        </element>
      </choice>
    </element>
  </choice>
</element>
```

Listing 9: Tasks Pattern RELAX NG representation

As we see from Listing 9 the scheme for this case contains a lot of `<choice>` and `<ref>` tags, some of which refer to their parent element (are recursive). It is hard to predict the constraints of binary

expression, which may only contain two simple task elements or other binary expressions, which may itself contain further complex expressions.

In the next editor window one can compose the responsibility patterns, which include one or more task patterns from first editor and one or more data elements. Listings 10 and 11 are demonstrating XML snippets and RELAX NG schema.

```
<responsibility>
  <binary type="and">
    <binaryoperand>
      <taskpattern>http://sumatra.pri.univie.ac.at/~demo/sprint/backend/tasks/
          patterns/3</taskpattern>
    </binaryoperand>
    <binaryoperand>
      <dataelement>dataA</dataelement>
    </binaryoperand>
  </binary>
</responsibility>
```

Listing 10: Responsibility Pattern XML representation

```
<element name="responsibility" xmlns="http://relaxng.org/ns/structure/1.0">
  <element name="binary">
    <attribute name="type">
      <text/>
    </attribute>
    <element name="binaryoperand">
      <choice>
        <element name="taskpattern">
          <text/>
        </element>
        <element name="dataelement">
          <text/>
        </element>
        <ref name="binary"/>
      </choice>
    </element>
    <ref name="binaryoperand"/>
  </element>
</element>
```

Listing 11: Responsibility Pattern RELAX NG representation

As you can see from XML listing, the task pattern element just holds the reference to existing task pattern on backend server.

In the constraints editor we make the pattern consisting of either data and task elements or only data or only task elements. Therefore the resulting XML structure may vary depending on target pattern. Listing 12 is demonstrating the mixed case with both types of elements.

```
<constraintspattern>
  <category>
    Control Flow
  </category>
  <operation>
    execute
  </operation>
  <constraints>
    <dependencies>
      <dependency>
        <binary>
```

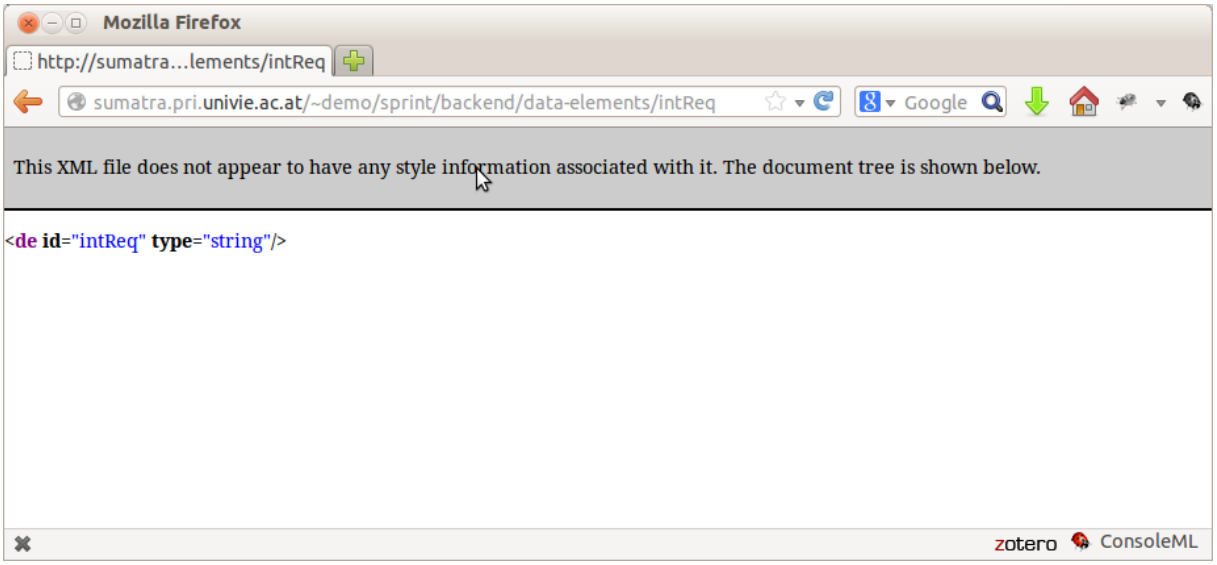


Figure 21: Backend Controller in Action

```
12            <binaryoperand>
13              <element>
14              http://sumatra.pri.univie.ac.at/~demo/sprint/backend/data-elements/
                    intReq
15              </element>
16            </binaryoperand>
17            <binaryoperand>
18            <element>
19            http://sumatra.pri.univie.ac.at/~demo/sprint/backend/tasks/a1
20            </element>
21            </binaryoperand>
22          </binary>
23        </dependency>
24      </dependencies>
25      <constraint id="pc0" type="time">
26      noon
27      </constraint>
28      <constraint id="pc1" type="location">
29      Lab A
30      </constraint>
31    </constraints>
32  </constraintspattern>
```

Listing 12: Constraints Pattern XML representation

as you see, the nodes, holding task and data elements are in fact the URLs referring to corresponding task/data on the server. Entering the URL in the browser window will display the element. That is achieved by means of using URL parser on the backend server, which is parsing the URL and generates response like in Fig. 21.

5 Test Cases and Evaluation

In the use cases we are going to show, how the use cases from section 4.1 are implemented using the editors we have created in the master thesis. We start with the first two use cases (1A and 1B) dealing

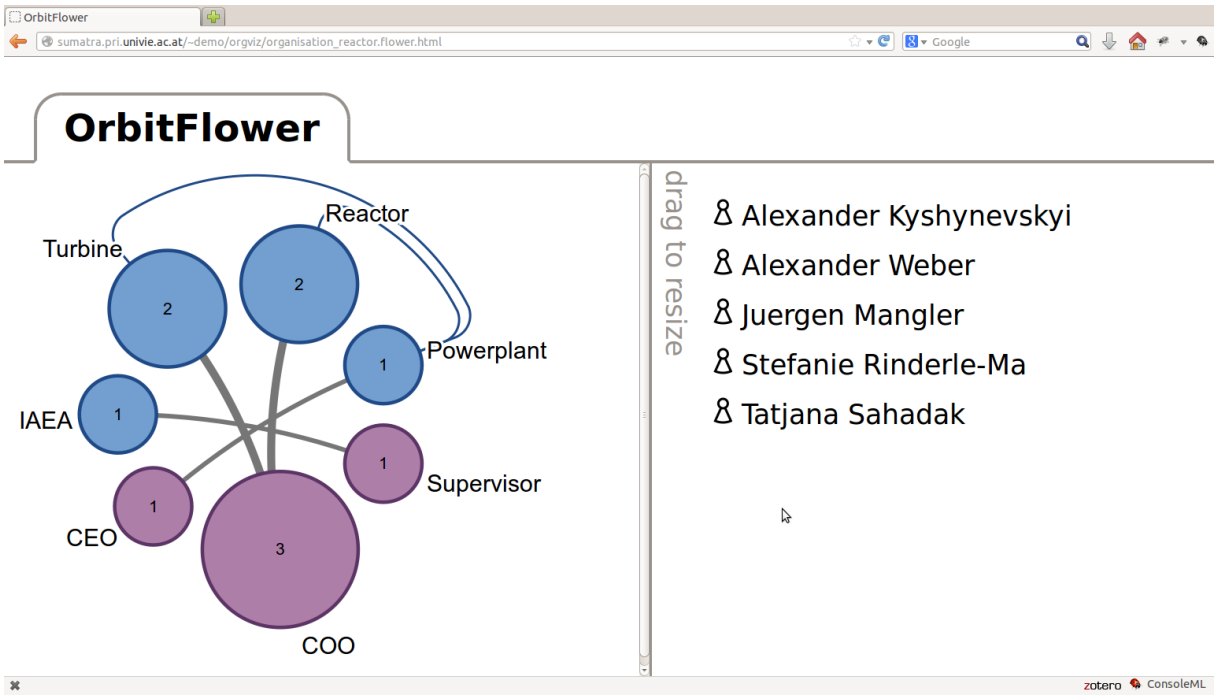


Figure 22: Orbit Flower Diagram of the NPP

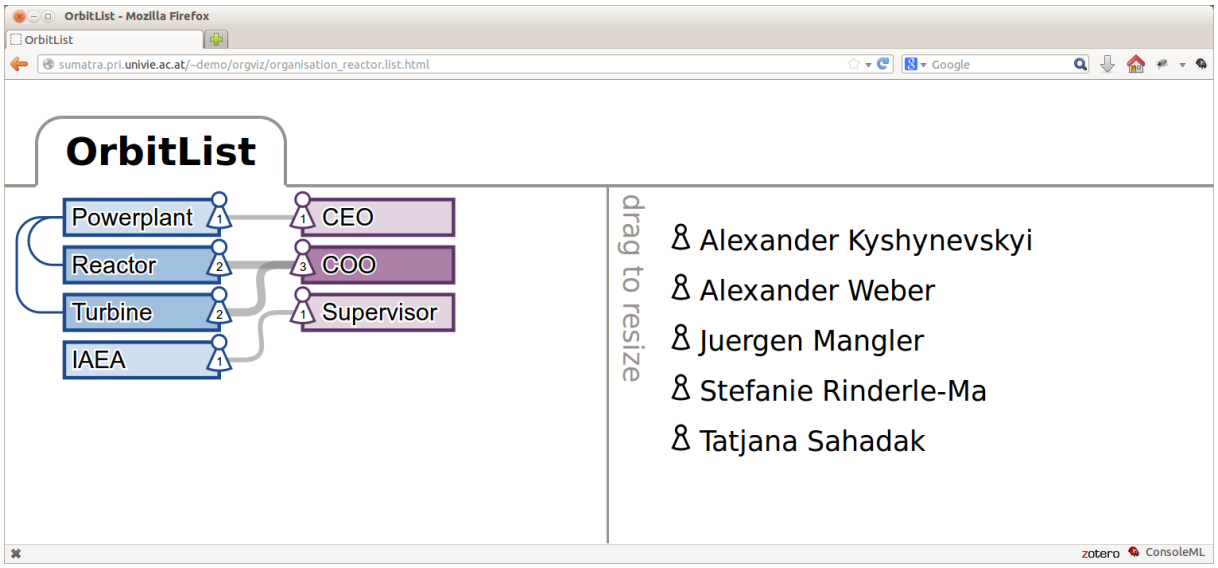


Figure 23: Orbit List Diagram of the NPP

with normal and unscheduled power reactor maintenance procedures. First of, we set up the application responsible for graphical representation of the organizational structure. That is achieved by writing appropriate XML files, which are visualized by the graphical interface (Fig. 22).

As can be seen from Fig. 22, it demonstrates all organizational units, their hierarchy (arches from Power plant to Reactor and Turbine), as well as roles and users. Another one representational approach is demonstrated at Fig. 23.

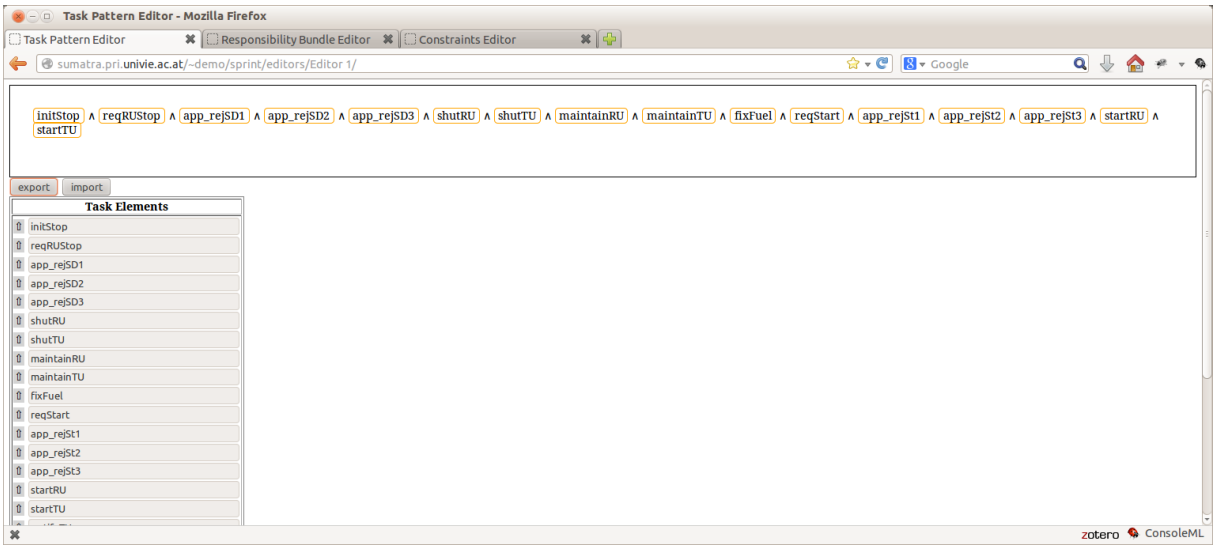


Figure 24: Use Case 1. Task Pattern $b_{\text{reactor maintenance}}$, $rc^{tp}_{1.4}$

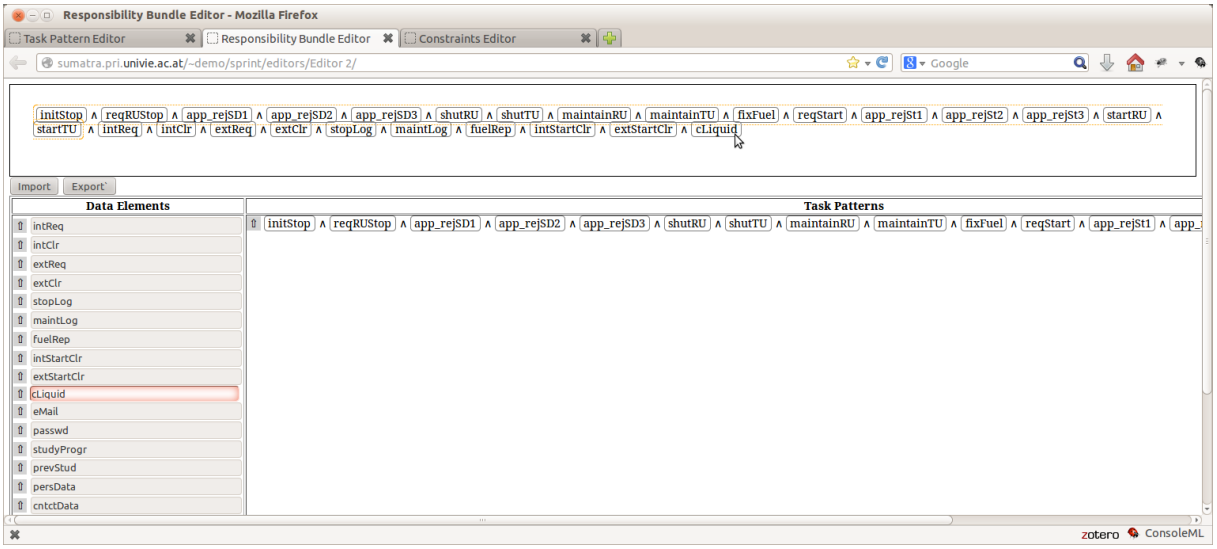


Figure 25: Use Case 1. Responsibility Pattern

The task pattern constraints from Editor 1 look as in Fig. 24.

The order of tasks doesn't play any role here. It only states, which tasks are allowed for the process. Please notice, that the task pattern constraint is not broken into three peaces as it was described in the document above. We omit that for the sake of compactness, but it can be easily made in the editor too. The task pattern is composed by means of dragging appropriate tasks onto upper formula area. Having made that user must export the pattern onto backend server by means of clicking the "Export" button. After that we are ready to compose the responsibility pattern including data elements and previously defined task pattern. The procedure is pretty much the same as for previous pattern (Fig. 25). We chose not to collapse the task pattern for this editor, as it would make the situation less understandable.

Constraints implementation is achieved by tweaking the JS classes for the third editor a tiny bit. That can be done with little effort. We will set it up as we go through constraints. The first constraint C_1 may

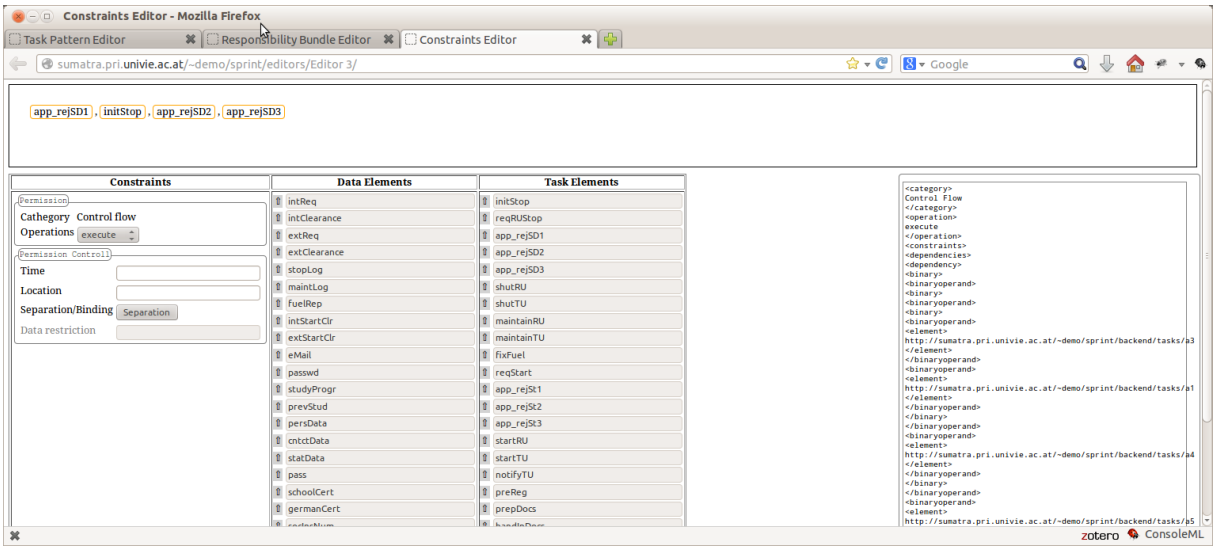


Figure 26: Use Case 1. Constraint C_1

be seen in Fig. 26. The user just needs to drag necessary elements over to the formula area and then click the toggable button ”Separation/Binding” till it gets appropriate label.

We should also notice, that the main part in the editor is in the `<div>` element with XML code in the right portion of the viewport. This piece of code holds all aspects of constraint being worked on. Listing 13 shows how C_1 is represented in that format.

```
<category>
  Control Flow
</category>
<operation>
  execute
</operation>
<constraints>
  <dependencies>
    <dependency>
    <binary>
      <binaryoperand>
      <binary>
      <binaryoperand>
        <binary>
          <binaryoperand>
            <element>
              http://sumatra.pri.univie.ac.at/~demo/sprint/backend/tasks/a3
            </element>
          </binaryoperand>
          <binaryoperand>
            <element>
              http://sumatra.pri.univie.ac.at/~demo/sprint/backend/tasks/a1
            </element>
          </binaryoperand>
        </binary>
      </binaryoperand>
      <binaryoperand>
        <element>
          http://sumatra.pri.univie.ac.at/~demo/sprint/backend/tasks/a4
```

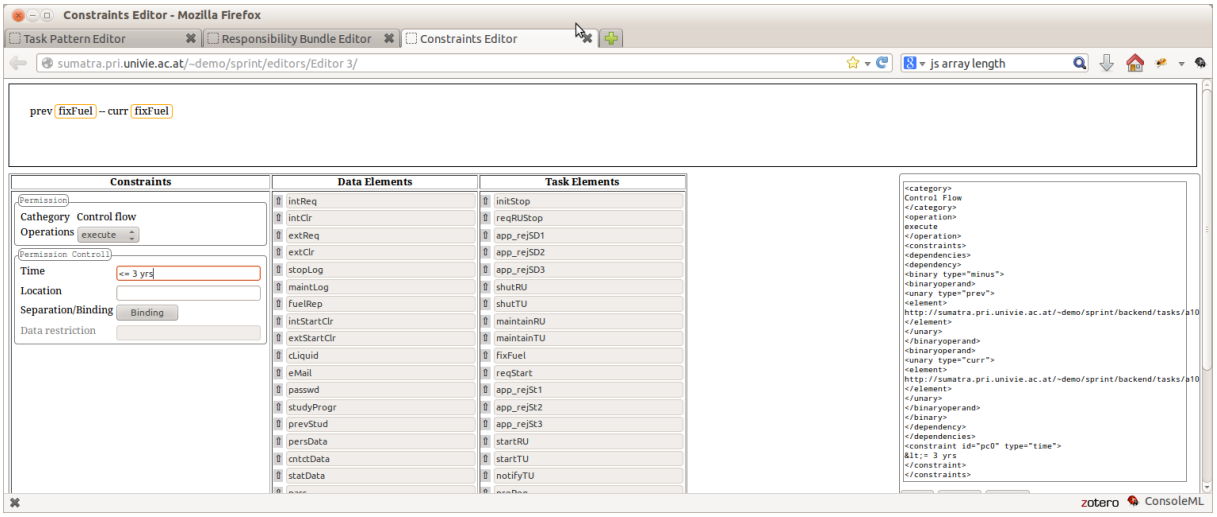


Figure 27: Use Case 1. Constraint C_2

```
30          </element>
31        </binaryoperand>
32        </binary>
33        </binaryoperand>
34        <binaryoperand>
35          <element>
36            http://sumatra.pri.univie.ac.at/~demo/sprint/backend/tasks/a5
37          </element>
38        </binaryoperand>
39      </binary>
40      </dependency>
41    </dependencies>
42    <constraint id="pc0" type="separationBinding">
43      separation
44    </constraint>
45  </constraints>
```

Listing 13: XML Constraint Representation

From this listing we can see that the elements are put together as operands of binary operator. We have got to know this format in previous section. Furthermore each element is represented by its URL rather than by the label which user can see in the formula area. Separation constraint is stored at the tail of the code.

We do the same in order to express, that shutdown and startup approvals/rejections from the side of IAEA should be issued by one and the same person (in our case Tanja Sahaidak). The only difference is that we impose binding constraint onto app_rejSD1 and app_rejSt1 tasks. Having showed these examples there is no need to demonstrate screen shots for each and every separation/binding constraint, as the process gets really straightforward.

In order to implement C_2 in the editor, there should be corresponding functions which return the data of last and current fuel manipulation procedures. These functions may be presented as unary operators in our case. That means all we have to do is to add the key-value pairs in the list of unary operators (Fig 27).

Constraint C_3 may be expressed in a few steps. We use here the same approach of subtracting time stamps as above (Fig. 28).

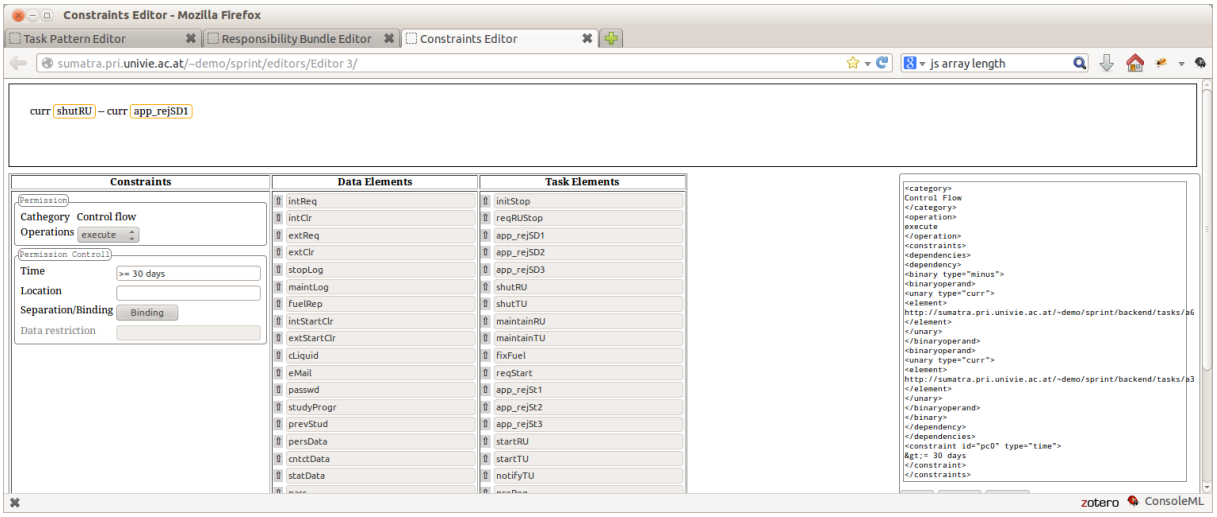


Figure 28: Use Case 1. Constraint C_3

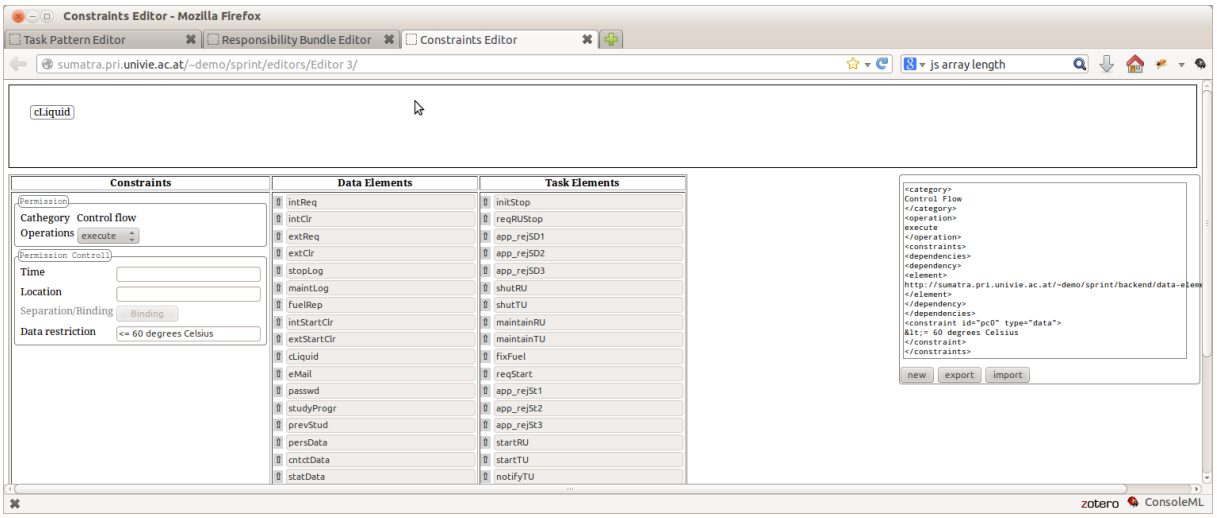


Figure 29: Use Case 1. Constraint C_5

In order to save place in the document, we won't place the screen shots for the second part of constraint C_3 and C_4, as they are implemented identically to what we just did.

For C_5 we use other input field in the editor, which corresponds to data-related content (Fig. 29).

In use case 2B, with contingency reactor shutdown, we are performing same steps as in scheduled shutdown use case, so there is no need to demonstrate that visually.

We act in the same manner for the bachelor and master enrollment use cases (use case 2A and 2B). First of we represent the organizational structure of the university (Fig. 30 and 31).

Then we compose the task pattern relation constraint (Fig. 32) for the case where all needed documents were handed in correctly.

The relation responsibility constraint is a bit bulky as we have quite a few data elements (Fig. 33)

As one can see, the task from previous use case are still present in the editor. Constraints C_2, C_3 and C_4 are time related and are all implemented in the same way, so we just place one screen shot here (Fig. 34).

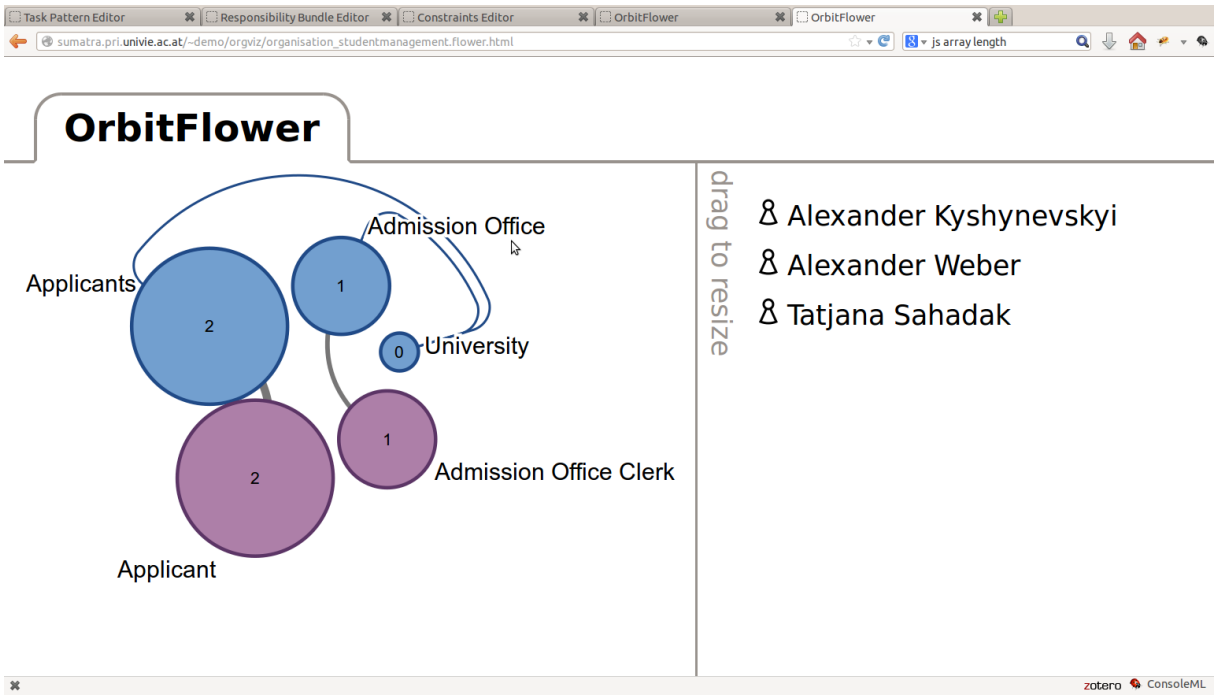


Figure 30: Use Case 2. Orbit Flower Diagram of the University

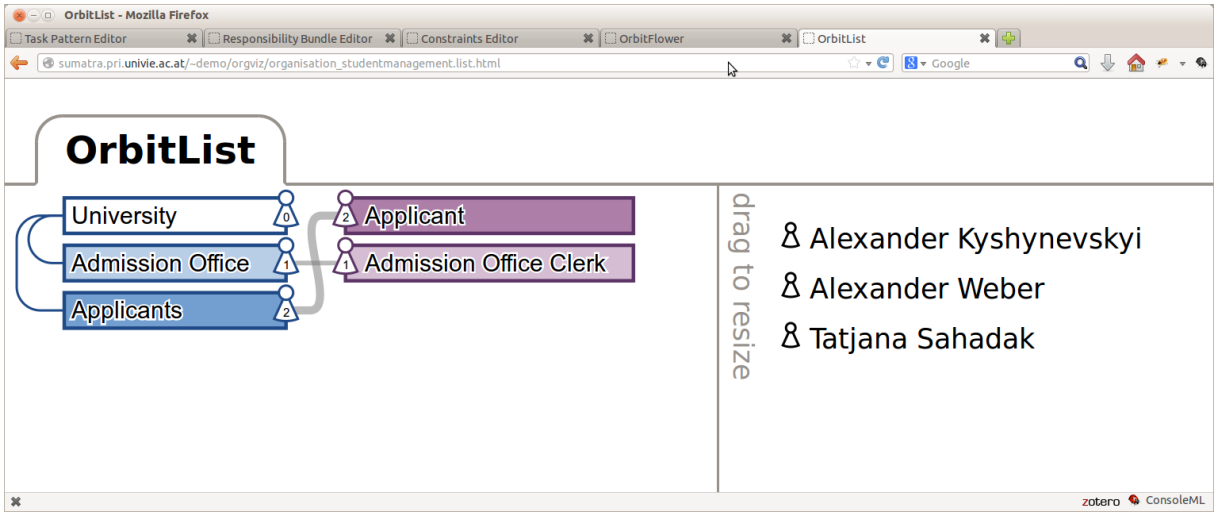


Figure 31: Use Case 2. Orbit List Diagram of the University

Constraint C_5 is simply a data restriction imposed onto data object (Fig. 35). In a real life there is no easy method to check the compliance with the requirement. So this restriction is rather symbolic.

C_6 is about German proficiency level and is similar to C_5. As for the last two constraints, we have represented them as regular expressions which in a real life may be parsed by the workflow engine. We only give one screen shot here as implementation of them in the editor is identical (Fig 36). The C_7 and C_8 are data object constraints as well and are implemented in the same way by putting the regular expression patterns in the ”Data restriction” field.

At the end we briefly represent task pattern (Fig. 37) and responsibility bundle (Fig. 38) for the use

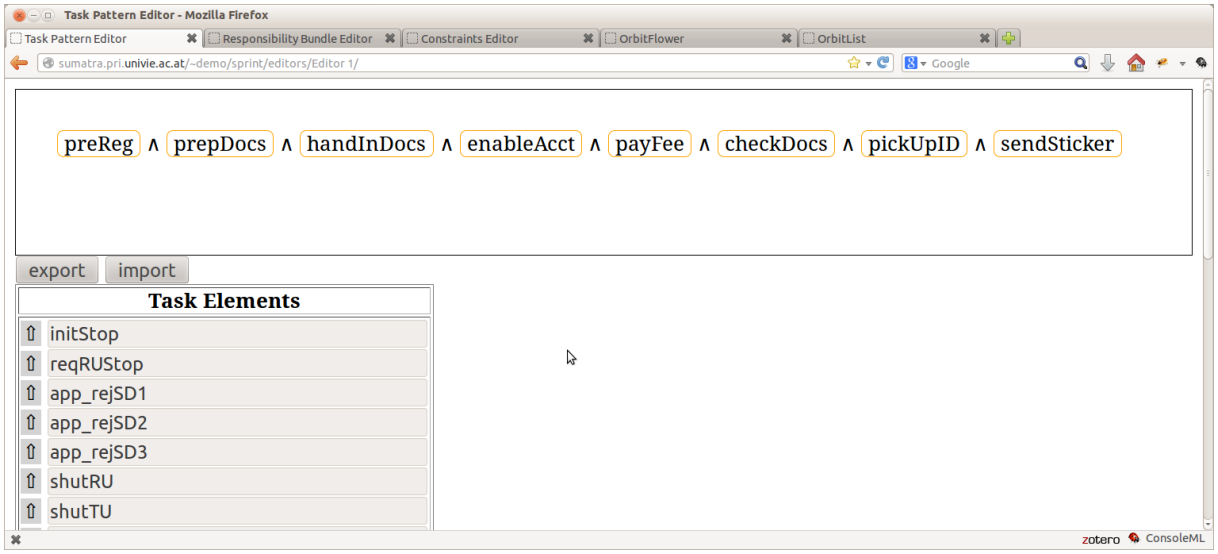


Figure 32: Use Case 2. Task Pattern Relation Constraint

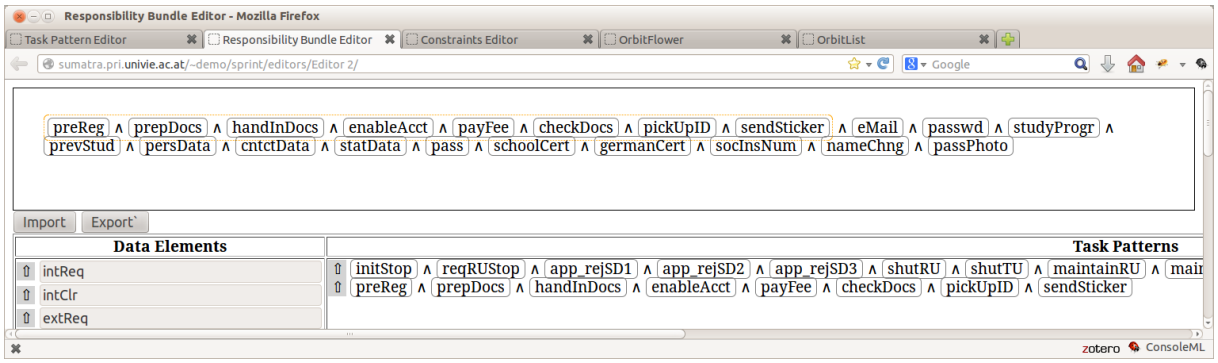


Figure 33: Use Case 2. Responsibility Relation Constraint

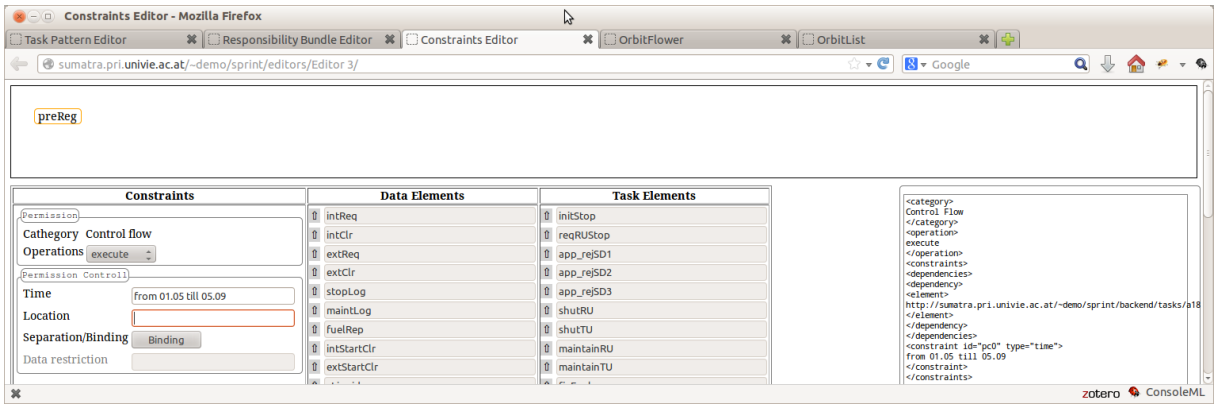


Figure 34: Use Case 2. Time Constraint C_2

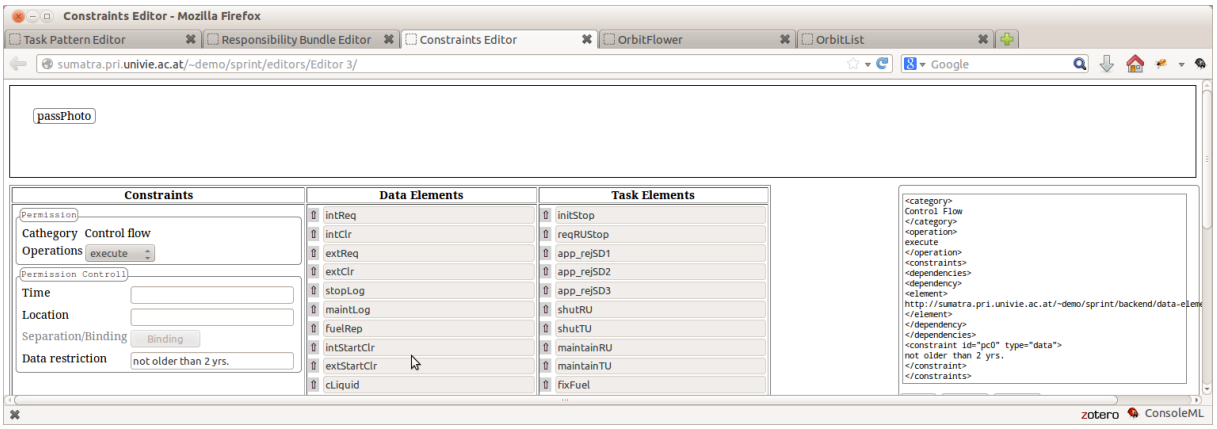


Figure 35: Use Case 2. Data Constraint C_5

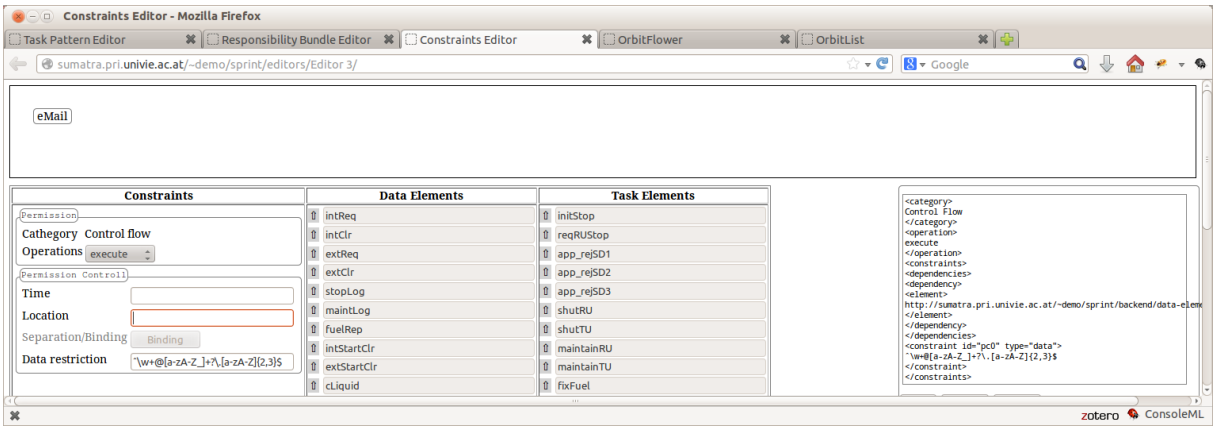


Figure 36: Use Case 2. Data Constraint C_7

case 2B, omitting the constraints as they are almost identical to what we have done in the use case 2A.

6 Conclusion and Summary

In this thesis we were working on design and development of the system aimed at supporting management processes of security policies of information systems. Those systems tend to grow in scale and get more complex over the time. As systems expand and grow, more and more actors, tasks and artifacts are involved in the business processes, which makes the security aspects hard to handle.

First we have analyzed the existing approaches, mainly how to define and representing small bundles of activities and related data (documents), called responsibilities. These bundles, in relation to permissions and roles are the basis for monitoring the design and execution of processes.

We then desribed the design and implementation of a system that allows for agile and flexible management of responsibilities in information systems. The underlying system supports composing task and responsibility patterns (which are combination of tasks and data elements). It also supports restriction for single elements (i.e. permissions) or patterns of elements (i.e. responsibility bundles). It should be noticed that the implemented system only handles the security aspects, whereas the organizational

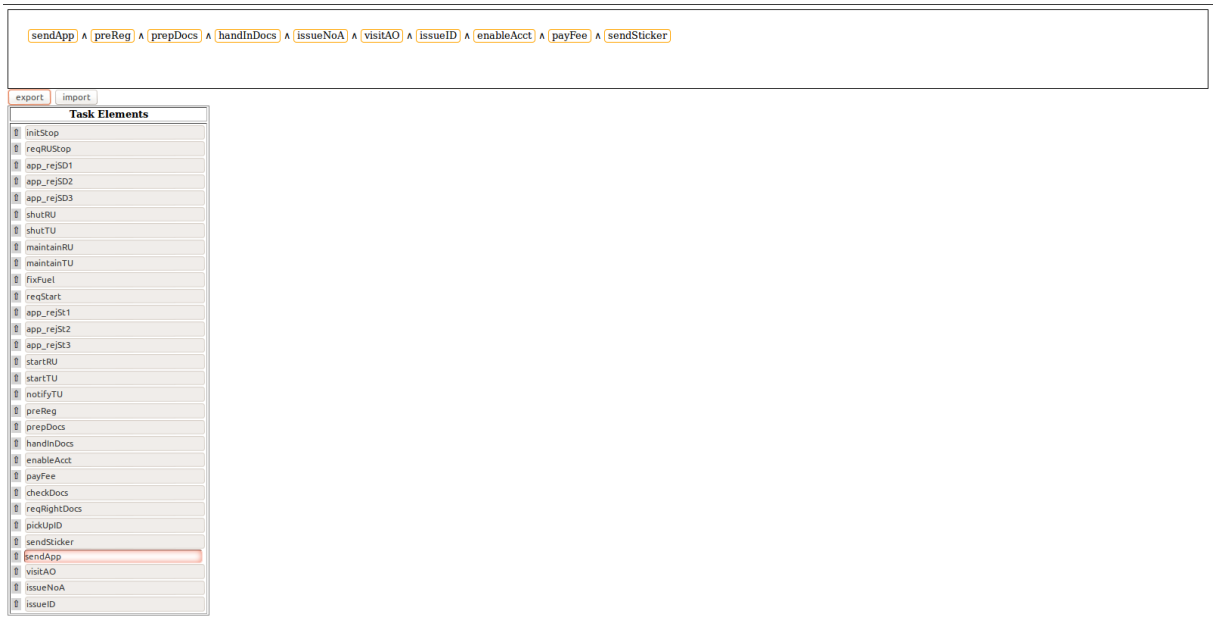


Figure 37: Use Case 2B: Task Pattern

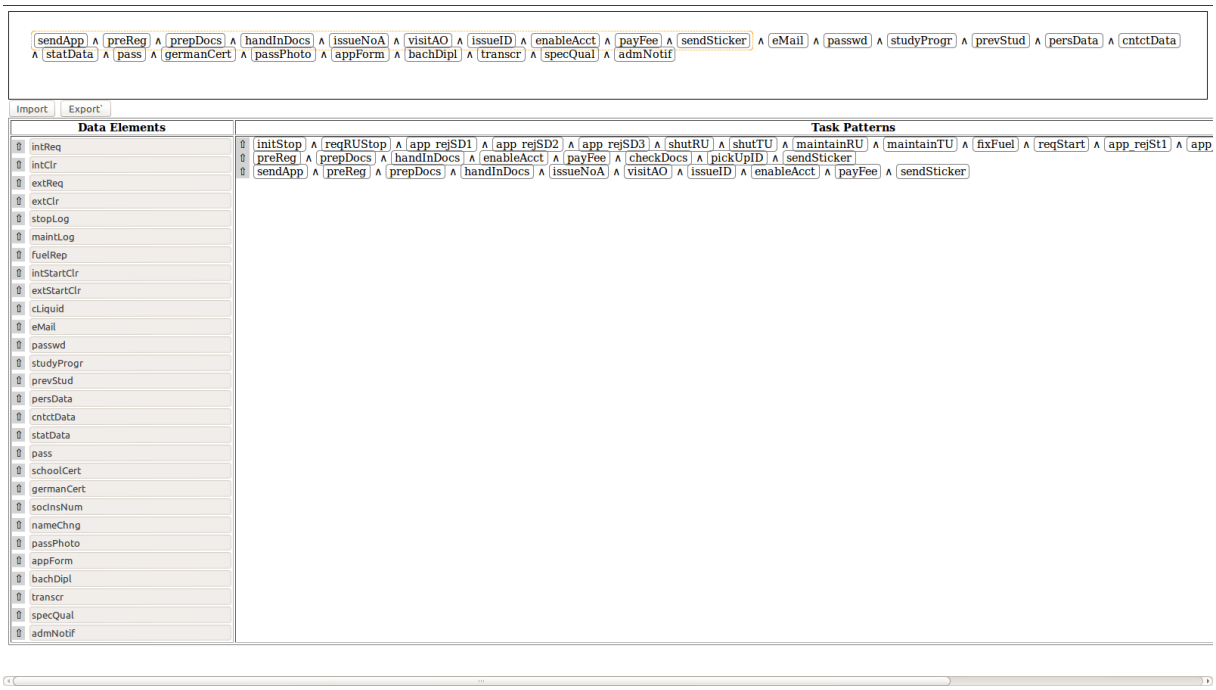


Figure 38: Use Case 2. Responsibility Pattern

part is taken care of by OrgViz [9], implemented in the Workflow Systems and Technology Group at the University of Vienna.

In order to evaluate the business process and implement security aspects, a couple of use cases were presented. In those use cases we looked at how structural as well as security information about enterprise can be extracted from textual description of the BP. That allows us to get the list of roles users and constraints. Finally the use cases were implemented using the OrgViz and the editor presented in this thesis.

At its current state the editor only allows to model the responsibilities and constraints in an organization, the future work may include combining the editor with the OrgViz by means of another user interface. Furthermore, connecting it to the workflow engine may provide further possibilities to directly monitor security aspects of the execution of BP instance. It will be also very important to develop a robust and reliable methodologies for security analysts who will be responsible for keeping the system running. In our cases we had to extract all constraints from a verbal description of the processes, separate data objects and tasks. For big and complex processes it is crucial to understand their scope and to be able not to miss critical constraints.

References

[1] I. Aedo, P. Díaz, and D. Sanz. An RBAC model-based approach to specify the access policies of web-based emergency information systems. *International Journal of Intelligent Control and Systems*, 11(4), 2006.

[2] C. P. Argyris. *R. & Smith, DM (1985) Action science: Concepts, Methods and Skills for Research and Intervention*. San Francisco: Jossey-Bass.

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

[4] M. Burch, F. Bott, F. Beck, and S. Diehl. Cartesian vs. radial–a comparative evaluation of two visualization tools. In *Advances in Visual Computing*, page 151–160. Springer, 2008.

[5] E. J. Coyne. Role engineering. In *Proceedings of the first ACM Workshop on Role-based access control*, page 4, 1996.

[6] E. A. Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, 995:1072, 1990.

[7] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001.

[8] P. Herrmann and G. Herrmann. Security requirement analysis of business processes. *Electronic Commerce Research*, 6(3-4):305–335, 2006.

[9] L. Hong, F. Meng, and J. Cai. Research on layout algorithms for better data visualization. In *Proc. of the 2nd Symposium Int'l Computer Science and Computational Technology (ISCSCT 09). Academy Publisher*, page 369–372, 2009.

[10] P. C. Hung and K. Karlapalem. A secure workflow model. In *Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003-Volume 21*, page 33–41, 2003.

[11] M. Jakl. Representational state transfer.

[12] S. Kriglstein, J. Mangler, and S. Rinderle-Ma. Who is who: On visualizing organizational models in collaborative systems. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on*, page 279–288, 2012.

[13] S. Kriglstein and G. Wallner. Human centered design in practice: A case study with the ontology visualization tool knoocks. In *Computer Vision, Imaging and Computer Graphics. Theory and Applications*, page 123–141. Springer, 2013.

[9] University of Vienna, http://sumatra.pri.univie.ac.at/~demo/orgviz/

[14] M. Kuhlmann, D. Shohat, and G. Schimpf. Role mining-revealing business roles for security administration using data mining technology. In *Proceedings of the eighth ACM symposium on Access control models and technologies*, page 179–186, 2003.

[15] O. Kulyk, R. Kosara, J. Urquiza, and I. Wassink. Human-centered aspects. In *Human-Centered Visualization Environments*, page 13–75. Springer, 2007.

[16] M. Leitner, J. Mangler, and S. Rinderle-Ma. SPRINT-Responsibilities: design and development of security policies in process-aware information systems. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, 2(4):4–26, 2011.

[17] D. A. Nadler and M. L. Tushman. A model for diagnosing organizational behavior. *Organizational Dynamics*, 9(2):35–51, 1980.

[18] G. Neumann and M. Strembeck. A scenario-driven role engineering process for functional RBAC roles. In *Proceedings of the seventh ACM symposium on Access control models and technologies*, page 33–42, 2002.

[19] M. Pesic and W. M. van der Aalst. A declarative approach for flexible business processes management. In *Business Process Management Workshops*, page 169–180, 2006.

[20] S. Rinderle and M. Reichert. A formal framework for adaptive access control models. In *Journal on data semantics IX*, page 82–112. Springer, 2007.

[21] S. Rinderle-Ma and M. Leitner. On evolving organizational models without losing control on authorization constraints in web service orchestrations. In *Commerce and Enterprise Computing (CEC), 2010 IEEE 12th Conference on*, page 128–135, 2010.

[22] R. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for role-based administration of roles. *ACM Transactions on Information and System Security*, 2(1):105–135, Feb. 1999.

[23] R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST model for role-based access control: towards a unified standard. In *ACM workshop on Role-based access control*, volume 2000, 2000.

[24] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, page 336–343, 1996.

[25] W. M. van der Aalst, H. A. Reijers, A. J. Weijters, B. F. van Dongen, A. K. Alves de Medeiros, M. Song, and H. M. W. Verbeek. Business process mining: An industrial application. *Information Systems*, 32(5):713–732, 2007.

[26] 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(04):455–485, 2003.

[27] B. Weber, M. Reichert, W. Wild, and S. Rinderle. Balancing flexibility and security in adaptive process management systems. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, page 59–76. Springer, 2005.