universität wien

MASTERARBEIT

Titel der Masterarbeit

"N2Sky - A Cloud-based Neural Network Simulation Environment"

verfasst von

Erwin Mann, BSc

Angestrebter akademischer Grad

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2013

Studienkennzahl lt. Studienblatt:	A 066 926
Studienrichtung lt. Studienblatt:	Masterstudium Wirtschaftsinformatik
Betreuer:	Univ.-Prof. Dipl.-Ing. Dr. techn. Erich Schikuta

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate deutlich kenntlich gemacht zu haben.

Wien, den 17. September 2013

Unterschrift:

_____

Erwin MANN

Acknowledgements

It gives me great pleasure in acknowledging the individual and personal support of my supervisor, Prof. Erich Schikuta. He inspired me with his analytical and critical skills and his patience whilst allowing me the room to work in my own way. Furthermore I would like to thank Wajeeha Khalil for her guidance through RAVO and Peter Paul Beran who gave me valuable technical notes on VINNSL and N2Grid.

Contents

Chapter 1

Introduction

1.1 Motivation

Understanding how the human brain works is one of the greatest challenges in science. The Human Brain Project [1] is an extremely ambitious attempt to answer this question by trying to create a realistic model of the human brain and simulate it on supercomputers based on latest findings in neuroscience. If future findings will continue to be as rich as now, achieving this goal might not seem as impossible as it seems today. One of the key tasks the research center in Jülich primarily responsible for is to develop a computing infrastructure which meets the immodest requirements: It has to be about a factor of 1000 faster than the most powerful supercomputers currently existing. This means to break through the border of Exaflop range (i.e. 10^{18} floating point operations per second).

Artificial neural networks are also inspired from human or animal central nervous system as systems of neurons interconnected by "synapses" (they are called *weights*) and are used typically to solve machine learning and pattern recognition tasks. Working with artificial neural networks is also very time-consuming and "resource-hungry". Neural network training and evaluation tasks cause the main part of the effort. Training tasks of neural networks containing large training data sets can take several hours or even days. Resources needed or created in this context should be accessible for the neural information processing community in a transparent way.

As described by the UK e-Science initiative several goals can be reached by the usage of new stimulating techniques [2]:

- Fostering more effective and seamless collaboration of dispersed scientific or commercial communities.
- Large-scale applications complying with these requirements should provide a uniform "look and feel" to a wide range of resources and location independence of data and computational resources.
- Enabling transparent access to "high-end" resources from the desktop.

Problem Statement. These considerations lead us to the following more specific problem description:

- Working with artificial neural networks needs elasticity concerning computing power as well as size and number of resources.
- Many scientists develop systems for their own neural network applications because existing programs often fail in providing a comprehensive, easy-to-use, and transparent interface,
- Most of these systems lack a generalized framework for handling data sets and neural networks homogeneously.
- It should be possible to unify different resources to one single resource and present it in a standardized way.
- Foster cooperation between members of scientific communities by sharing these resources.

These problems leads to the basic idea of the present Master's thesis, namely to create a neural network simulation environment consisting of services in the Clouds that can be securely accessed from everywhere.

The layout of this Master's thesis is as follows: In Chapter 2 the N2Sky architecture is described in general whereas Chapter 3 presents every N2Sky component in detail., The interface of N2Sky is laid out in Chapter 4. The tutorial in Chapter 5 describes how to create, train and evaluate a simple *Backpropagation* neural network The *User Guide* in Chapter 6 is divided into two parts: *End User Guide* (6.1) and *Administraton Guide* (6.2). Chapter 7 includes a *Developer's Guide* The thesis ends with a look at further developments and research directions in Chapter 8.

1.2 Terms and Definitions

An **Artificial Neural Network** (ANN) or in short a Neural Network (NN) is an information processing paradigm that is inspired by the way biological central nervous systems, such as the brain, process information. Hecht-Nielsen, the inventor of one of the first neurocomputers, defines an ANN as "a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs" [3]. An introduction to Computational Intelligence in general and Neural Networks in detail can be found in [4].

A **Virtual Organization** (VO) is a logical orchestration of globally dispersed resources to achieve common goals [5]. A VO couples a wide variety of geographically distributed computational resources (such as PCs, workstations and supercomputers), databases, storage systems, libraries and special purpose scientific instruments. This wide range of resources then is present as one integrated unified resource. Once prepared, such resources can be provided to communities. In the Computational Intelligence community

these advancements in sharing resources transparently are not used to the maximum possible extent until now.

For the term **Reference Architecture (RA)** different definitions exist The one which is important in our context is presented by IEEE standard making institute. An RA is defined as a way of documenting good architecture design practice to address commonly occurring problem [6]. Muller defines a list of criteria for a good RA [7]:

- Understandable for a broad set of heterogeneous stakeholders like customers, product managers, project managers or engineers,
- Accessible and actually read or seen by a majority of the organization,
- Addresses the key issues of the specific domain,
- Satisfactory quality,
- Acceptable,
- Up-to-date and maintainable and
- Adds value to the business.

N2Sky is an artificial neural network simulation environment providing basic functions like creating, training and evaluating neural networks and is presented in this thesis. The system is Cloud-based in order to allow for a growing user community. Our simulator interacts with Cloud data resources (i.e. databases) to store and retrieve all relevant data about static and dynamic components of NN objects. Cloud computing resources provide elastic processing cycles for "power-hungry" NN simulations. N2Sky provides a standardized description language for describing neural net paradigms and objects called VINNSL [8] and a business model for researchers and students but also for any interested customer. Furthermore our system allows to be extended by additional NN paradigms provided by arbitrary developers.

The term *N2Sky* was inspired by the emerging *Sky* computing model described in [9]. **Sky computing** was defined as an architectural concept that denotes federated Cloud computing. It allows for the creation of large infrastructures consisting of Clouds of different affinity, i.e. providing different types of resources, e.g. computational power, disk space, networks, etc., which work together to form one giant Cloud or, as it were, a sky computer.

During the last years many experts have tried to define the term **Cloud Compting** [10], [11]. For our purpose, Buyya's definition appears most appropriate: "A Cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualised computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers" [12]. This recalls the definition of Virtual Organizations and emphasizes the existence of a business model which is one of the main differences between Cloud Computing and Grid Computing. A more technical one is provided by the U.S. National Institute of Standards and Technology where the essential characteristics of Cloud Computing is to enable ubiquitous, convenient, on-

demand network access to a shared pool of configurable computing resources like servers, networks, storage, services and applications. Such resources can be rapidly provisioned and released with low system administration effort or minimal service provider interaction [13].

1.3 Related work

Related projects can be classified into two categories: *Virtual Organizations for Computational Intelligence* and *Artificial Neural Network Simulators*. The first one focuses on community building and resource sharing, whereas the second one subsumes only a small selection of neural network simulation engines already existing.

1.3.1 Virtual Organizations for Computational Intelligence

In the field of Computational Intelligence and Machine Learning we were able to identify only two existing VOs: RAVO and CIML.

RAVO. The **Reference Architecture for Virtual Organizations (RAVO) [14]** was developed by Wajeeha Khalil, a PhD student at the University of Vienna. It is presented as a standard for building Virtual Organizations (VOs). It gives a starting point for electronic collaboration in one or more domains for developers, organizations and individuals RAVO consists of two parts.

1. **The guidance for requirements analysis.** In the first phase boundaries of the VO are defined and mandatory and optional components are identified. In case of evolution of an existing system to a VO, additionally a *Gap analysis* is also performed.
2. **The blueprint for a layered architecture.** The **N2Sky architecture** presented in Chapter 2 is derived from RAVO.

CIML. The CIML portal [15] is an international multi-university initiative to form a VO in the field of Computational Intelligence and Machine Learning. Its primary purpose is to help facilitate a virtual scientific community infrastructure for all those involved with or interested in this field. The CIML portal provides both research and education resources, and specific application-oriented resources residing at the portal or are linked from the CIML site.

The CIML portal is, besides its goal to gather all type of users interested in Computational Intelligence and Machine Learning. It provides only a static pool of knowledge resources and totally lacks other forms of resources and new computing paradigms for supporting collaborative work. Furthermore it is built without clear design principles. Its environment is built using a conventional approach and without giving clear information about the employed IT infrastructure. Therefore the CIML community doesn't have the necessary motivation to contribute and we can say that the acceptance of the CIML portal is relatively low.

1.3.2 Artificial Neural Network Simulators

In the last two decades a large number of neural network simulators have been developed. The following presents only a few of them.

SOM-PAK. The **Self-Organizing Map Program Package (SOM-PAK)** [16] was developed in the early 1990s at the Helsinki University of Technology. It defines the SOM as "a mapping from the input data space R^n onto a regular two-dimensional array of nodes" [16]. In other words we can say that the SOM is a "nonlinear projection" of the probability density function $p(x)$ of the high-dimensional input data onto the two-dimensional display like a flower that is pressed. SOM-PAK provides a set of programs that run separately from the console.

SNNS. The **Stuttgart Neural Network Simulator (SNNS)** [17] was developed at the Institute for Parallel and Distributed High Performance Systems at the University of Stuttgart since 1989. It consists of four main components:

1. Simulator kernel,
2. Graphical user interface,
3. Batch execution interface *batchman* and
4. Network compiler *snns2c*

The GUI can be used to create, manipulate and visualize neural nets directly in various ways, so it should also be well suited for unexperienced users. An important design concept is to enable the user to select only those aspects of the visual representation of the neural net in which he is interested.

Over the last few years several artificial neural network simulation systems were developed at the University of Vienna fostering up-to-date computer science paradigms:

1. **NeuroAccess.** NeuroAccess [18] was developed in 1998 and deals with the conceptual and physical integration of neural networks into relational database systems.
2. **NeuroWeb.** NeuroWeb [19] is an artificial neural network simulator which exploits Internet-based networks as a transparent layer to exchange information (neural network paradigms and neural network objects). NeuroWeb was presented in 2002.
3. **N2Grid.** The N2Grid system [20] was developed in 2004 reusing several proven design principles from NeuroWeb. The original idea behind N2Grid was to consider all components of an artificial neural network as data objects that can be serialized and stored at some data site in the Grid,
4. **N2Cloud.** The N2Cloud system [21] is a further evolution step of N2Grid and was presented in 2010. It is based on a service-oriented architecture (SOA) and will use storage services provided by a Cloud environment.
5. **N2Sky.** The presented N2Sky environment takes up the technology of N2Cloud to a new dimension using RAVO and the virtual organization paradigm.

Chapter 2

The N2Sky Architecture

In this chapter we present a requirements analysis consisting of a questionnaire and a list of functional requirements, as well as the resulting N2Sky architecture designed to make use of Cloud technology and service-orientation.

2.1 Requirements Analysis

2.1.1 Questionnaire

This updated and extended questionnaire is based on the N2Sky case study [14] drawn up in common with Wajeeha Khalil. The first six questions (Q1-Q6) are derived from phase 1 of RAVO's requirements analysis as they are:

- "**Q1:** Why to form a VO? What are the reasons of an organization to create a VO?
- **Q2:** What is the motivation behind participation? Why should other persons, institutes, service providers want to participate in a VO?
- **Q3:** What services are offered by a VO?
- **Q4:** How are these services fared? What is the type of the resources/business model?
- **Q5:** Who are the intended users? Who will eventually use and get benefited from this VO?
- **Q6:** What is the life of (membership of) a VO? Are temporal alliance or permanent participation expected?" [14].

Question Q7 refers to phase 2 (Identification of Components) whereas Q8 to Q10 initiate a Gap analysis. According to the RAVO guidance for requirements analysis this additional analysis is required because N2Sky is not a completely new system but rather is also based on previous developments.

2.1.1.1 The need for a VO

Question Q1:

What are the reasons to form a Virtual Organization for N2sky?

Answer:

This question can be answered keeping two aspects in mind: technical needs and social aspects. N2Sky is based on a Cloud infrastructure instead of a Grid infrastructure as provided by N2Grid. For **technical aspects** five Cloud characteristics [13] can be revisited to develop a VO for the neural network community:

1. **Shared Pool of Resources.** Resources are shared by multiple tenants. A tenant is defined by the type of Cloud being used. Therefore a tenant can be either a department, organization, institution, etc. N2Sky shares besides hardware resources also knowledge resources. This allows the creation of a shared pool of neural net paradigms, neural net objects and other data and information between researchers, end users and developers worldwide.

2. **On-demand self-service.** Consumers can create their computing resources (server, operating system, or software) within mere minutes of deciding they need it without requiring human interaction with each service provider. N2Sky allows for transparent access to "high-end" resources (computing and knowledge resources) stored within the Cloud on a global scale from desktop or smart phone, i.e. whenever the consumer needs is provided independently from the local infrastructure situation.

3. **Broad network access.** Users can access computing resources from anywhere they need it as long as they are connected to the network. N2Sky fosters location independence of computational, storage and network resources.

4. **Rapid elasticity.** Computing resources can scale up or scale down based on the users needs. To end users this appears to be unlimited resources. N2Sky delivers to the users a resource infrastructure which scales according to the problem. This leads to the situation that always the necessary resources are available to provide efficient and standardized solutions for any neural network problem.

5. **Measured service.** Services of Cloud systems automatically control and optimize resource use by leveraging a metering capability enabling the pay-as-you-go model. This allows consumers of computing resources to pay based on their use of the resource. N2Sky supports the creation of neural network business models. Access to neural network resources, as novel paradigms or trained neural nets for specific problem solutions. Resources can be offered for free or following certain business regulations, e.g. a certain fee for usage or access only for specific user groups.

"For **social aspects** forming a VO is to bring the people together who are common in some respect. Sometimes goals unite the people and sometimes problems bring them closer. So building a community with group of people having problems and those who have solutions can be achieved in form of a VO. A trusted platform to share their com-

monalities in terms of knowledge, information, applications and procedures makes the face of a VO well recognized and accepted." [14]. Besides neural network simulation methods, N2Sky's main focus is on sharing resources among the neural network community. Creating a VO helps in forming this community in a structured way.

2.1.1.2 The motivation behind participation

Question Q2:

What is the motivation behind participation? Why should other persons, institutes, service providers want to participate in N2Sky's VO?

Answer:

"This is the key question, which discovers the needs of participating entities in a VO thus defining the problem domain. Identification of common needs has an important impact on the shape of a VO" [14]. Some specific reasons for joining our N2Sky community are listed as follows:

- "Share neural net paradigms, neural net objects and other data and information between researchers, developers and end users worldwide.
- Provide for an efficient and standardized solutions to NN problems.
- Transparent access to "high-end" neural resources stored within the Cloud from desktop or smart phone.
- Provide a uniform "look and feel" to NN resources.
- Location independence of computational, storage and network resources" [14].
- Still no standard exists defining a description language for the exchange of resources among artificial neural network simulators. VINNSL [8] is a proposal for such a specification.
- Existing neural network simulation systems "lack of a generalized framework for handling data sets and neural networks homogenously. During the training phase and the evaluation phase of a neural net the user has to feed the net with large amounts of data. Conventionally data sets are mostly supported via sequential files only and the definition of the input stream, output or target stream of a neural net is often static and extremely complex" [20].
- The launch of N2Sky may inspire other researchers to collaborate and share their individual resources.
- Creation, training and evaluation of a great number of neural net resources on a free basis.

A use case in the problem field of cancer cell classification is described in Subsection 2.3.2.

2.1.1.3 Provided services

Question Q3:

What services are offered by N2Sky's VO? Which scenarios are supported by N2Sky?

Answer:

Below there is a partial list of supported scenarios including stakeholders involved in them:

- "Read and discuss turorials, documentations, presentations and research papers (All stakeholders).
- Publish turorials, research papers and presentations (Contributors).
- Manage stakeholder account: edit data, select payment method, credit account, cash out (All stakeholders).
- Integrate hardware into N2Sky, e.g. scanner or sensors and select pricing models to offer (Contributors).
- Simulation services: Create, train and evaluate neural objects (Consumers).
- Create end user bill: calculate and send bill, debit amount and credit parts of it the contributors (Controller as subclass of Administrator).
- Check stakeholder accounts and send reminders if bills were not paid (Controllers)" [14].
- Upload input sample data per copy/paste or via database queries (Consumers).
- Upload new paradigm services to N2Sky and choose appropriate pricing model (Contributors).
- Integrate paradigms after a successfully passed upload (Administrators).
- Query Interface: Query proven solution approaches of the community, currently i.e. search for neural network problems and their solutions (Consumers).
- Select resources that will be used and choose a pricing model for them (Consumers).

2.1.1.4 Business model

Question Q4:

How are these services fared? What is the type of the resources/business model? Is there a specific business model as foreseen in RAVO?

Answer:

The goal of our business model is to cover the greatest possible number of different customer and provider requirements. Currently N2Sky is used in an educational context

and offers its services fro free. After implementing additional security concepts we plan to offer N2Sky also in a commercial context. We plan to implement following pricing models where the total price will be the sum of paradigm provider fees and Cloud provider fees in three different SLAs:

1. **Pay-per-use.** Paradigm usage in three service level variants: *Premium, Standard* and *Minimal*.
2. **Flat rate.** The user is charged with the pice P for the selected SLA for m months with a limit of x computation cycles per month.
3. **Local execution.** For experienced users it is also possible to download paradigm/s to his own machine (notebook, workstation or server) so he has to pay only for those paradigm/s and a small manipulation fee to the Cloud provider.
4. **Negotiated roles.** Large customers who have special requirements are given the possibility to agree to special conditions which can then be reused for similar clients.
5. **Dynamic negotiation.** If the customer wants to use the N2Sky system extensively it is possible to dynamically negotiate conditions both with the Cloud provider and the paradigm vendor where these negotiations should be moderated by the N2Sky system.

A detailed description of N2sky business model and SLA possibilities is provided in Subsection 3.2.3.

2.1.1.5 Intended VO users

Question Q5:

Who are the intended users? Which stakeholders will eventually use and get benefited from N2Sky's VO?

Answer:

These following groups of stakeholders will eventually get benefited from N2Sky. In Subsection 3.2.5 these roles are described in detail.

- **Subjects.** Subjects are both contributors and consumers, e.g. neural net paradigm providers and paradigm end users. Possible examples for subjects are neural network researchers, professors, Master's and PhD students.
- **End users.** Users only consuming but not providing services. Possible examples are researchers, lecturers, students and commercial users that are interested in neural network problem solutions
- **Developer**. Paradigm service developer contribute services and provide it for End Users either for free or choosing a particular pricing model on a pay-per-use basis, monthly flat rates or special agreements.

- **Administrators.** Administrators can be either system administrators (N2Sky Administrator) or business administrators (N2Sky Controller). The N2Sky controller controls the business workflow (management of resources and pricing models, payments, invoicing, bookkeeping, reminders),

2.1.1.6 Duration of the N2Sky VO

Question Q6:

What is the life of the VO? Is permanent participation expected or only temporal alliance?

Answer:

N2Sky is not created for a specific period of time. It is open for updates and improvements from authenticated stakeholders. In an educational context at the university attendees will participate normally for the duration of the course, i.e. 4-5 months but are invited to extend their membership for further contribution.

2.1.1.7 Components identified

Question Q7:

Which building blocks are missing in N2Sky in comparison to RAVO and CIML VOs?

Answer:

We can identify two missing components for N2Sky: Workflow tools, Provenance and User authorization:

- **Workflow tools.** To execute micro flows during neural network training and evaluation phases it should be possible to integrate several workflow tools. Workflow systems are described in **Subsection** 3.2.7.
- **Provenance.** To collect metadata for each neural network simulation task it should be possible to integrate data provenance tools.
- **User authorization:** Sharing resources in a secure way needs to integrate user authorization methods. Access control is described in Subsection 3.2.6.

2.1.1.8 Gap Analysis

Question Q8:

What are the existing VOs in the field of Computational Intelligence and why you based your development on RAVO?

Answer:

The two existing VOs, RAVO and CIML are characterized in Subsection 1.3.1. I choosed RAVO's architecture because it identifies components within a Cloud SPI stack consisting

of five layers. It helped me in identifying and appending components for a special VO in the field of neural networks. I think RAVO is a better architectural concept than CIML in establishing VOs because it is optimized to run on Cloud infrastructures. Nevertheless a collaboration with CIML is also quite conceivable.

S.NO	FEATURES	N2Grid [20]	N2Cloud [21]	N2Sky
A	PORTAL INFORMATION			
1	Introduction	Y	Y	Y
2	Team members	Y	Y	Y
3	Tutorials	Y	Y	Y
4	News/Updates	Y	Y	Y
5	FAQs	Y	Y	Y
6	Contact	Y	Y	Y
B	RESOURCE INFORMATION			
1	Glossary	Y	Y	Y
2	Publications	Y	Y	Y
3	Presentations	Y	Y	Y
4	Links to related resources	Y	Y	Y
5	Software/Applications	Y	Y	Y
6	Info on techn. architecture	Y	Y	Y
7	Newsletter	N	N	N
C	RESOURCES			
1	Calendar	Y	Y	Y
2	Blogs/Forum	Y	Y	Y
3	Research communities/Sites	Y	Y	Y
4	Workshops/Seminars	Y	Y	Y
5	Chat/Email	N	N	N
D	USER INFORMATION			
1	Sign in	N	N	Y
2	Create account	N	N	Y
3	Forgot your password	N	N	N
E	TASK INFORMATION			
1	Data search	N	N	N
2	Graphical result display	Y	Y	Y
3	Adding services	Y	Y	Y
4	Portal usage statistics	N	N	N
5	Tools	N	N	N
6	Workflow tools	N	N	N
7	Provenance tools	N	N	N

Table 2.1: Gap Analysis - Comparing N2Sky Virtual Organization's characteristics to its predecessors N2Grid and N2Cloud.[1]

[1]Source: Own formation as continuation of Table 5.2 in [14] where N2Grid is compared with existing VO characteristics in the field of Computation Intelligence and Machine Learning.

Question Q9:

On which system is N2Sky an upgrade?

Answer:

N2Sky is based on previous developments at the Workflow Systems and Technology Group. The *Simulation Management* component and *Simulation Services* are further developments of the *N2Grid Service* whereas interface specifications are based on the *Neural network specification language VINNSL*.

Table 2.1 compares N2sky's VO characteristics with its predecessors N2Grid and N2cloud classified in five main components identified in [7]: *Portal information, Resource information, Resources, User information* and *Task information.* In contrast to *N2Grid* and *N2Cloud* N2Sky contains user information, especially a sign-in mechanism and an account registration form for registering new users.

Question Q10:

Which logical and physical resources are available in N2Sky's resource hierarchy? Was RAVO's resource categorization applicable to it?

Answer:

Computation and storage are categorized according to RAVO's physical resources whereas network traffic is appended to our physical resource hierarchy. Logical resources are derived from RAVO and are refined especially for neural network purposes. We developed a detailed resource hierarchy described in Subsection 3.2.5. Resources can be bundled to resource groups (e.g. grouping several paradigm variants to a set of paradigms which can be chosen or subscribed all at once.).

2.1.1.9 Implementation

Question Q11:

Which technologies do you chose for implementing N2sky components? Please justify your decisions in brief.

Answer:

N2Sky services running on servers:

- **Java.** N2sky server components are implemented in Java because N2Grid is also Java-based
- **Spring framework.** The Spring framework is a lightweight alternative to EJBs to realize service-based software components. Used features of Spring are Spring Beans and JPA to implement our domain model.
- **Tomcat.** The Spring Framework only needs a Java web application container. We use Tomcat as application server but N2Sky will run on any JVM-compatible application server.

- **Maven2.** Project management and build tasks are supported by Maven2. A key factor for Maven is dependency management. Dependencies (Java libraries) and plugins (adapters to external software) can be managed and were downloaded from repositories automatically (configured in `pom.xml`).
- **Jersey.** We chose Jersey as framework for realizing RESTful Web Services because its production-ready quality and its large developer community. Jersey is the reference implementation for RESTful services provided by Sun/Oracle.

N2sky Mobile Web Portal:

- **HTML5.** The *Mobile Web Portal* is pure HTML5, CSS3 and JavaScript so iOS or Android apps can be created over PhoneGap with minimal additional effort.
- **jQuery Mobile.** As user interface design framework jQuery Mobile (jQM) is used because it is JavaScript and CSS-based by providing a standardized way of using stylesheets and UI elements like buttons, lists or input fields.
- **The-M-Project.** As jQM-based MVC framework *The-M-Project* is used to separate code into three categories: *Model, View* and *Controller*.

Cloud Infrastructure:

- **Eucalyptus:** The freely available Eucalyptus is used to create private Clouds at our own infrastructure. It is compatible with Amazon Web Services (EC2 and S3).
- **AWS.** As testing environment we use Amazon Web Services because a variety of operating systems and data stores are provided just ready to run.

2.1.2 Functional Requirements

The following functional requirements formalize and complete requirements not occured in answers of the questionnaire presented above.

2.1.2.1 SOA

1. Every component of N2Sky should be available as a Service.
2. Components within the SaaS should be browser-accessible or be downloadable as desktop or smart phone apps.
3. Components within the PaaS are connected by Web Services or APIs
4. Components within the IaaS should be accessible via Web Services or protocols.

2.1.2.2 Cloud Computing

1. All N2Sky components can easily be integrated into a Cloud computing infrastructure like Eucalyptus or Amazon EC2.

2. It should be possible to integrate a variety of relationsl DBMS to persist N2Sky's domain model.

3. Each instance on a Cloud-based node should have access to our relational DBMS, if more than one RDBMS are running within Clouds, they should be synchronized automatically.

4. It should be possible to integrate a variety of key/value stores and document-based stores like MongoDB. Synchronization between multiple stores is not required.

5. Every authorized user should be able to query any persistent object within the Cloud.

6. A fine-grained access control system should control the visibility of shared resources depending on stakeholder roles and project memberships (only for commercial use).

7. Elasticity in terms of computing and storage resources.

2.1.2.3 Neural Networks

1. Many different neural network paradigms can be offered for one theoretical problem.

2. A developer should be able to easily integrate a newly developed neural network paradigm into N2Sky.

2.1.2.4 User Interface

1. A uniform "look and feel" should be provided to neural network resources.

2. The web applications should also be accessible by smart phone browsers.

3. A Query interface like Google should enable users to search for theoretical problems and matching their existing solution methods.

4. A simple and intuitive providing of paradigm services is needed to get it for free or to purchase it ("iTunes for paradigms")

2.1.2.5 Business Model

1. Developers should be given the opportunity to earn money with their neural network paradigms.

2. End users should be able to choose between different pricing models.

2.2 The N2Sky Architecture as an Instance of RAVO

The basic categorization of the N2Sky architecture depends on the three categories of the Cloud computing SPI stack as they are: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS).

In the scientific community Cloud computing is sometimes stated as the natural evolution of Grid computing. Cloud computing therefore became an awesome word after IBM and Google collaborated in using this computing paradigm followed by the launch of IBM's *Blue Cloud* [22].

The N2Sky architecture is derived from the RAVO architecture [14] which is depicted in Figure 2.1 on the left hand side. The main difference to the standard SPI stack is that PaaS and IaaS each are further divided into two layers. The diagram on the right hand side in Figure **??** depicts the N2Sky architecture at the end of the first integration phase whereas the extended model as goal of the second integration phase is shown in Figure 2.2 (yellow components: mandatory, white: phase 1, grey: phase 2). During the preparation of this thesis mainly mandatory components were implemented respectively integrated. Integrating the other components remains for further work.

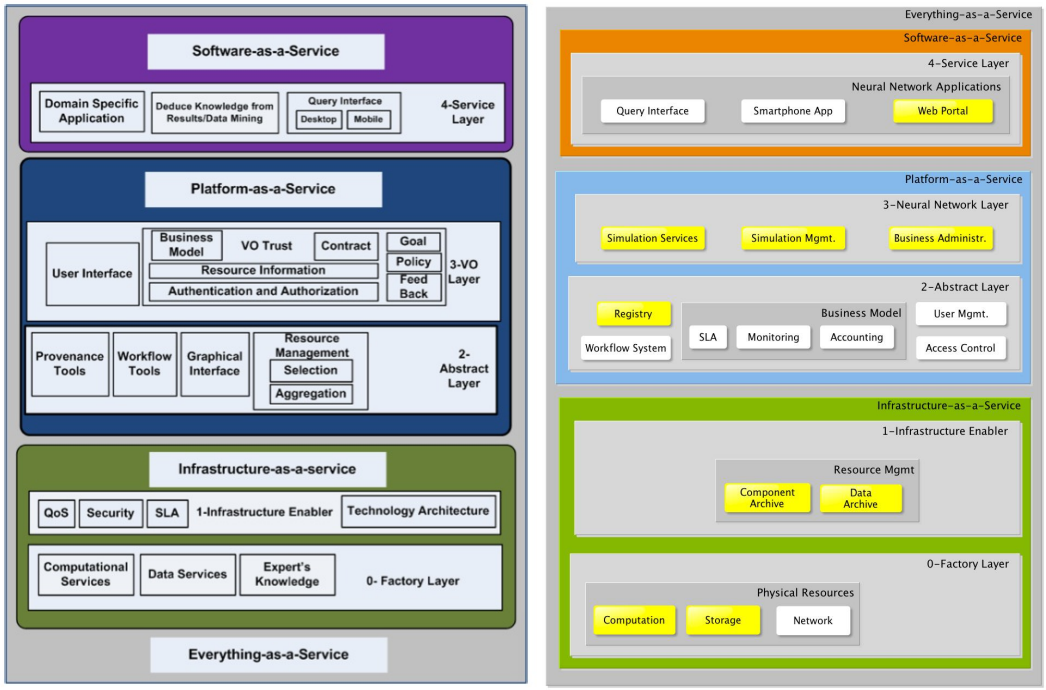

Figure 2.1: The RAVO architecture and the derived N2Sky architecture model after integration phase 1.[2]

[2]Source of diagram on the left: [14, p. 43].

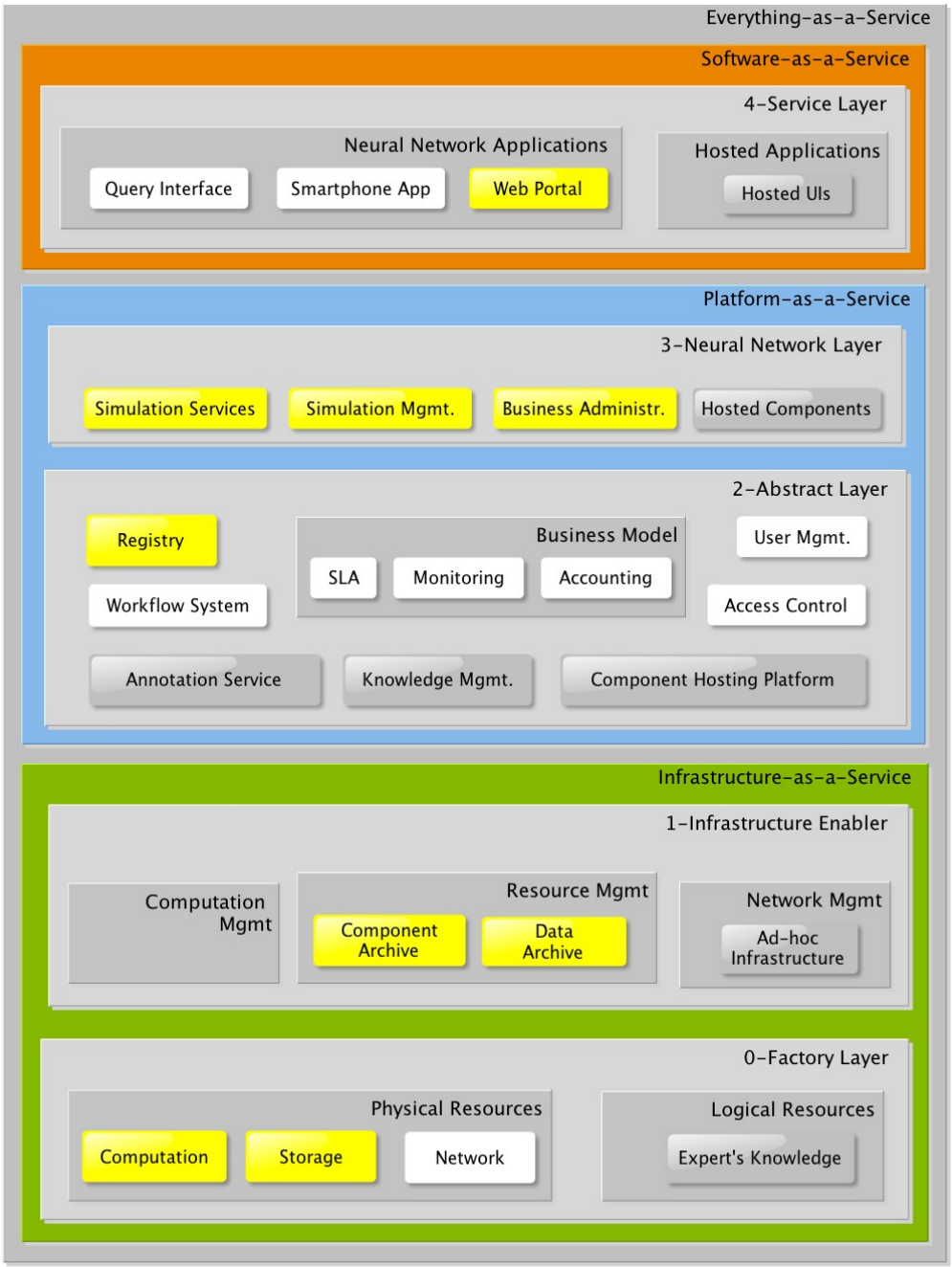

Figure 2.2: The N2Sky architecture after integration phase 2.

2.2.1 Infrastructure as a Service (IaaS)

The IaaS layer is all about managing resources, IaaS "basically provides enhanced virtualisation capabilities. Accordingly, different resources may be provided via a service interface" [23]. In the N2Sky architecture the IaaS layer consists of two sublayers: *Factory Layer* and *Infrastructure Enabler Layer.* Users need administrative rights for accessing the resources in Layer 0 over the resource management services in Layer 1.

- **Layer 0.** The **Factory Layer** contains physical and logical resources for N2Sky components. Physical resources comprise of hardware devices for storage, computation cycles and network traffic in a distributed manner. Logical resources contain expert's knowledge helping solving special problems like the *Paradigm Matching problem* described in Subsection 3.4.1.
- **Layer 1.** The **Infrastructure Enabler Layer** allows access to resources provided by the Factory layer. It consists of protocols, procedures and methods to manage the desired resources.

2.2.2 Platform as a Service (PaaS)

PaaS is all about application or service hosting on an abstract or more domain-specific basis. PaaS provides "computational resources via a platform upon which applications and services can be developed and hosted. PaaS typically makes use of dedicated APIs to control the behaviour of a server hosting engine which executes and replicates the execution according to user requests" [23]. It provides transparent access to resources offered by the IaaS layer and transparent access for applications offered by the SaaS layer. The Google App Engine, Force.com and Windows Azure are the most common examples. In the N2Sky architecture it is divided into two sublayers:

- **Layer 2.** The **Abstract Layer** contains domain-independent tools that are designed not only for use in connection with neural networks. A slight difference to RAVO is that we see service level agreements (SLAs) as part of our business model instead of placing it directly in Layer 1.
- **Layer 3.** The **Neural Network Layer** is composed of domain-specific (i.e. neural network-specific) applications.

2.2.3 Software as a Service (SaaS)

Finally, the SaaS layer on top of the SPI stack consists of Cloud-enabled ready-to-use applications or services, Saas offers "implementations of specific business functions and business processes that are provided with specific Cloud capabilities, i.e. they provide applications / services using a Cloud infrastructure or platform, rather than providing Cloud features themselves" [23]. Common examples are Google Docs, Microsoft Office 365, SAP Business by Design or Salesforce CRM. In context of N2Sky, SaaS is composed of one layer, namely the *Service Layer*.

- **Layer 4.** The **Service Layer** contains the user interfaces of applications provided in Layer 3 and is an entry point for both end users and contributors. Components are hosted in the Cloud or can be downloaded to local workstations or mobile devices.

2.2.4 Everything as a Service (XaaS)

Each of the five layers provide its functionality in a pure service-oriented manner so we can say that the N2Sky architecture realizes Everything-as-a-Service.

Table 2.2 again shows the complete list of N2Sky components or technologies including references to their component and interface descriptions, each assigned to one layer of the extended SPI stack.

	Section Architect.	Section Interface	Service L. (4)	Neur. N. L. (3)	Abstr. L. (2)	Infr. En. L. (1)
Web Portal	3.4.2	4.4.2	x			
smart phone App	3.4.3	4.4.3	x			
Query Interface	3.4.1	4.4.1	x			
Hosted UIs	3.4.4	4.4.4	x			
Simulation Services	3.3.1	4.3.1		x		
Simulation Management	3.3.2	4.3.2		x		
Business Administration	3.3.3	4.3.3		x		
Hosted Components	3.3.4	4.3.4		x		
Comp. Hosting Platform	3.2.9	4.2.9			x	
Registry	3.2.1	4.2.1			x	
Monitoring	3.2.2	4.2.2			x	
SLA	3.2.3	4.2.3			x	
Controlling and Acclunting	3.2.4	4.2.4			x	
Usermanagement	3.2.5	4.2.5			x	
Access Control	3.2.6	4.2.6			x	
Workflow System	3.2.7	4.2.7			x	
Knowledge Management	3.2.8	4.2.8			x	
Annotation Service	3.2.10	4.2.10			x	
Component Archive	3.1.1	4.1.1				x
Data Archive	3.1.2	4.1.2				x
Ad-hoc Infrastructure	3.1.3	4.1.3				x

Table 2.2: Components with corresponding layer-assignments.

2.3 Scenarios

2.3.1 A Sample Workflow

Before we present N2Sky components in detail we now focus on component collaboration. Figure 2.3 depicts an overall sample workflow consisting of mandatory and phase-1-components. The numbers refer to the labels in the diagram.

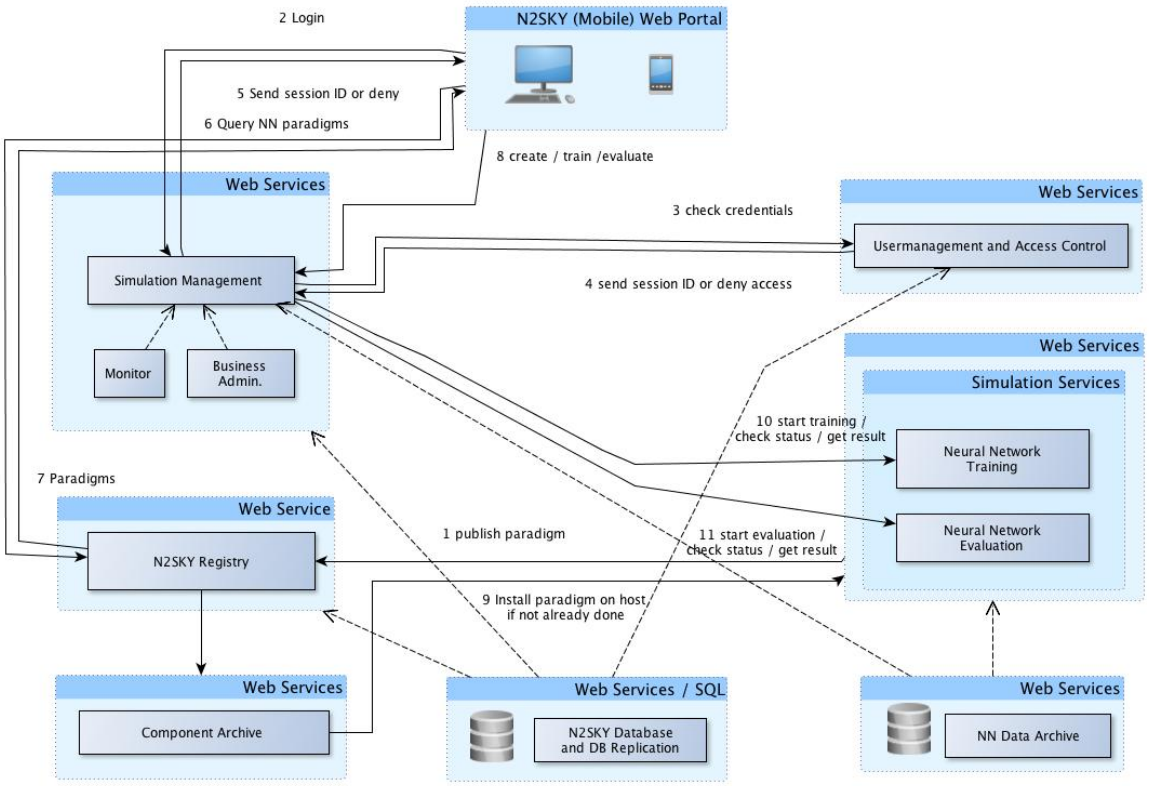

Figure 2.3: A N2Sky sample workflow.

1. The developer publishesa paradigm service to N2Sky as described in Subsection 7.2.
2. Stakeholder login via (mobile) web browser (AJAX request, RESTful Web Service).
3. *Simulation management* service dispatches login request to *User management and access control* component per RESTful Web Service.
4. Callback to *Simulation management* service either sending a new session id or deny access.
5. Callback to (mobile) web browser, redirecting session id or deny access.
6. Query *Registry* for neural network paradigms for problem solving.
7. Callback to (mobile) web browser by sending paradigm metadata.

8. Create new neural object by using selected paradigm for free, start new Eucalyptus node instance if needed, start training and after them start a new evaluation by using training result. The stakeholder view of the Simulation Core Process is described in detail in Subsection 3.3.2.

9. Before a training task is able to start properly, it is checked if the desired paradigm is provided at this host. If not, a Java EE web archive is deployed to this host by retrieving it from the *Component archive* service.

10. Start a new training thread - *Simulation management* checks training status periodically until status == 100, then get result and store it over data archive in database.

11. Start a new evaluation thread - *Simulation management* checks evaluation status periodically until status == 100, then get result and store it over data archive in database.

2.3.2 N2Sky Use Case: Cancer Cell Classification

The following use case based on the *BrainMaker Breast Cancer Cell Classification* project [24] demonstrates interactions between N2Sky components in real life. The professional environment: A team of artificial neural network programmers and cancer researcher develop a system to search for breast cancer cells in tissue images made by an electron microscope. Breast cancer diagnoses are created automatically from an artificial neural network-enabled system on the basis of classified cancer cells. The following N2Sky components are involved:

- **Hosted UI component.** In contrast to the referenced *BrainMaker* project we assume that the user interface component is integrated into a rich client application at a desktop computer in a hospital whereas the cancer diagnosis service component is hosted in the Cloud on the *Neural Network Layer*. This makes sense because the rich client application profits from an elaborated operation-system-specific user interface whereas the "power-hungry" neural network component runs within an elastic Cloud.

- **Component Hosting Platform.** This platform is based on an Eucalyptus Cloud infrastructure and provides an easy-to-use method to integrate hosted components like the *Breast cancer cell classification service*. We assume that this component is a Spring-based Java web application deployable on Java web application containers (in this case Tomcat 6). The domain model is persisted in a RDBMS (in this case PostgreSQL) over a JPA-based O/R mapping.

- **Simulation Management Service.** The *Cancer cell classification component* calls neural net simulation methods like `createNeuralObject,() uploadData(), train()` *and* `evaluate()` by passing JSON data over RESTful Web Services. The simulation management component itself calls standardized methods like `train(), evaluate(), checkStatus(), getResult()` of a simulation service in the same way.

- **Simulation Service.** A Backpropagation paradigm service is used for training and evaluation of images shot by an electron microscope. This service is a standard image-processing neural network solving various classification problems.
- **Business Administration Service.** The Cost Controller selects a pricing model (pay-per-use, service level agreement (SLA level B), debits money, adds user accounts for employees working for this project and sets budget limits for each employee. At the other side, the N2Sky Controller performs billing, payment and reminder cycles.
- **Registry.** The registry provides information about each available component and checks every 30 minutes if it is still available. If not, the related component is removed automatically from the service list.
- **Business Model.** The customer decided to choose a pay-per-use model on SLA level B for hosting, computation and paradigm usage. The main part of these revenues will be forwarded to the paradigm service provider, the other part remains with the N2Sky hoster to cover operating cost. The SLA level B was chosen because the customer wants to save money and accepts a possibly slower computation at peak times. A detailed description of pricing models is provided in Subsection 3.2.3.
- **Usermanagement.** Each End User involved in analyzing electron microscope images gets an individual N2Sky login. The Head of Institute also gets the Cost Controller role because he is the project manager and therefore cost bearer and has the opportunity of setting budget limits for each user within this project.
- **Access Control.** Only the pre-selected Backpropagation paradigm service is accessible for the End User. To use more paradigms, the contract between End User and Cost Controller has to be modified and results to higher costs. Thereby the Cost Controller has to confirm or deny this contract.
- **Annotation Service.** It is possible to annotate images with various medical information This metadata then is displayed as an additional UI element as part of the user interface.
- **Component Archive.** If the simulation service is not available at the current Cloud node it will be retrieved from the component archive as `.WAR` file ready to run on Tomcat. Multiple Cloud nodes are running by using automated load balancing.
- **Data Archive.** Input and output data is stored and retrieved from a document-based store, in our case, MongoDB. The *Data archive* is responsible for this.
- **Ad-hoc Infrastructure.** The electron microscope as an additional special instrument is integrated into the image analyzing system enabling interaction with each other as simple as possible.

Chapter 3

Components of N2Sky

This chapter presents a detailed deescription of each N2Sky component according to the architecture model of implementation phase 2 depicted in Figure 2.2.

3.1 IaaS: Infrastructure Enabler

3.1.1 Component Archive

The *Component archive* is the central source for storing and retrieving components, especially neural network paradigm services. Within a Cloud environmentt the *Component archive* runs on exactly one single instance, otherwise additional technologies in the field of Distributed Systems will be required. Component replication is essential in cases of dynamically adding further components and simulation parallelization to accelerate it. The BPMN diagram in Figure 3.1 depicts the dynamic component replication process where the *Component archive* checks if a replica of a desired paradigm service is located on a desired host. If not, it will be replicated by copying it from the archive. Only after this householding activities training and evaluation tasks can be initiated on this host.

3.1.2 Data Archive

The *Data archive* component is used for storing and retrieving neural network-specific resources in JSON or XML format. When using OGSA-DAI [25] as data resource management interface data sources could be either file systems or various RDBMS or XML database systems. The *N2Sky Data archive* is derived from this concept, authorized user are able to store and retrieve JSON data to or from key/value stores or document-based datastores over RESTful Web Services. We are currently using this approach because OGSA-DAI does not support key/value and document-based stores in the current version 4.2. *Data archive* queries are described in detail in Subsection 6.1.3.

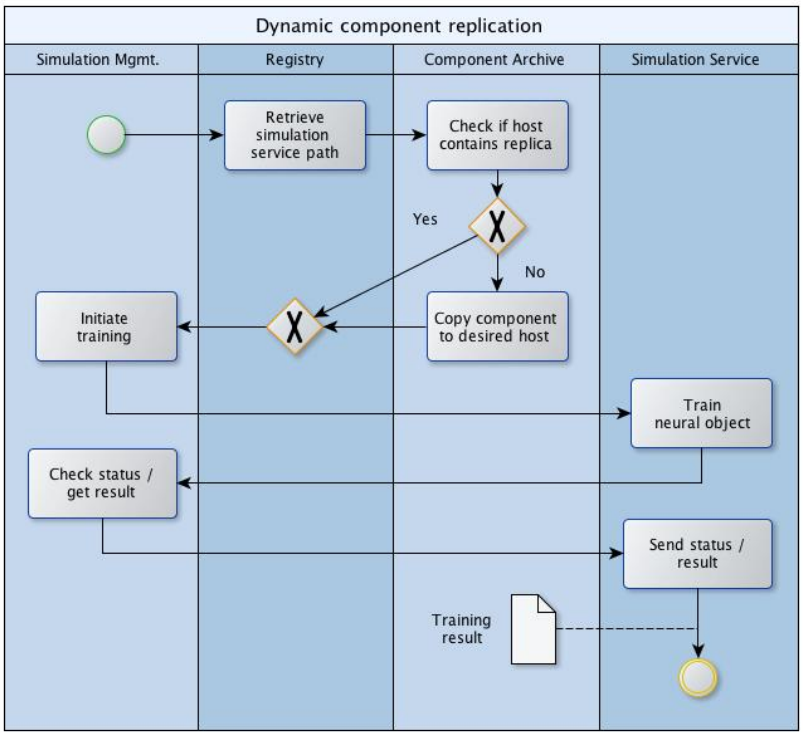

Figure 3.1: Dynamic component replication

3.1.3 Ad-hoc Infrastructure

Given the growing importance of smart phones and other mobile computing devices in our business world there is an increasing need for networks which can be assembled dynamically. Suppose a scientist attends a conference at a foreign university and wants to use his own smart phone and laptop.to communicate with other conference participants over N2Sky or over a conference management system. Traditionally the university provides computer terminals and WLAN access for free and without having to enter a password. Although this is just simple for the organization team, but has some drawbacks:

- Anyone can access the system, even those who are outside of the building.
- System usage is free, but is already included in the attendance fee.
- In many cases, terminal computers only provide internet acces and some basic applications, so that participants prefer to use their own equipment.
- If one's phone is used, high roaming fees are charged by the mobile operator.

Would it be not practical if the phone would automatically detect that WLAN is available and logs in as a device of a conference participant into an encrypted network? A personalized and time-limited password can easily be obtained by e-mail via a conference management system. In this case a secure connection will be established and unauthorized persons have no access to the network. Participants are also entitled to share printers, scanners and other peripherals.

In a larger context, a cafe can offer its IT infrastructure to its guests so they are able to achieve high bandwidth by using their own mobile devices. The coffee house owner could take advantage of this offer not only to increase its number of guests, but also earn money. A price list makes costs transparent and the user can specify a cost limit, to be sure not to receive a surprisingly high bill. Payment may be made via the mobile phone invoice or via a special transfer system as that in N2Sky.

3.2 PaaS: Abstract Layer

3.2.1 Registry

The *Registry* is a central component of the N2Sky architecture as it allows other components to find any desired service within the Cloud. Each involved service registers at this component to get known to other components.

Key functions of the *Registry* are:

- First-time registration of new services.
- Periodically checks if a registered service is still available. If not, it will be deregistered automatically.
- Get information about accepted XML/JSON Schema files
- Get information about stored attributes for each registered service
- Search for services by those stored attributes

At starting time the N2Sky system checks for each service held in the management database (`Resource` entities in our domain model) if it is still available. After a positive check, a service will be registered and after this it will be available for components in SaaS and PaaS layers. Components in SaaS layer are able to search for services by stored attributes which can be also queried. Each registered service gets an ID and is rgistered for a certain time (it is called `holdbacktime`, i.e. a configurable number of seconds). Before it is elapsed, the registry triggers a further availability check and only if this check is negative, the service will be removed from registry. In case of a positive check the time of the next check will be set to `currentTime + holdbackTime` as listed in Subsection 4.2.1.

3.2.2 Monitoring

The *Monitoring* component should provide methods for two different types of monitoring [26]:

1. **Business Activity Monitoring (BAM) [27].** Business Activity Monitoring collects business data and metrics like Key Performance Indicators (KPI) [28] as a basis of business management decisions.

2. **Technical or IT Monitoring.** It is concentrated on monitoring technical aspects of an IT system and helps IT professionals in their system administration tasks. IT monitoring includes usage and performance statistics as well as capacity and resource planning tasks.

In this context we identified key performance indicators determining success or failure of a neural net paradigm:

- The accuracy of evaluation results in relation to the time the training task consumed.
- Needed time effort to prepare input data for the chosen paradigm.
- The needed time effort to interpret evaluation results.
- The average number of stars customers have given for a paradigm after testing it.
- The monthly revenue made by providing a paradigm in N2Sky.

3.2.3 SLA

The N2Grid system was used so far only within the research community in a non-commercial way. To ensure a fair use and to offer the possibility to achieve profits for Cloud providers and neural network paradigm developers we propose a business model for N2Sky. Before it can function properly, the security model has to be implemented. Table 3.1. shows several pricing models for users of N2Sky.

Pricing model	Cloud provider SLA			Paradigm vendor
	A	B	C	
Premium	xp_A	-	-	$\sum_{i=1}^{n} y_i q_i$
Standard	xp_B	xp_B	-	$\sum_{i=1}^{n} y_i q_i$
Minimal	-	xp_C	xp_C	$\sum_{i=1}^{n} y_i q_i$
Flat rate	$P_{XA}m$	$P_{XB}m$	$P_{XC}m$	$\sum_{i=1}^{n} mQ_i$
Local execution	0	0	0	$\sum_{i=1}^{n} mQ_i$
Negotiated roles	P_{XAN}	P_{XBN}	P_{XCN}	Q_{YN}
Dynamic negotiat.	$f(x, p_A)$	$f(x, p_B)$	$f(x, p_C)$	$f(y, q)$

Table 3.1: N2Sky pricing models.

User fees are subdivided into two parts, the infrastructure of the Cloud provider and usage of neural network paradigms. To offer an incentive to develop computing-efficient paradigms these two parts are calculated separately instead of revenue sharing with predefined percentage rates. Cloud provider fees are dependent on processing cycles x and the price per cycle p which on its part is dependent on the chosen *Service level agreement (SLA)*. We differentiate between three service levels:

- **Service level A.** Simulation requests will be executed preferentially by reserving a particular percentage of the system resources for these requests.

- **Service level B.** Simulation requests will be executed in parallel by reserving another particular percentage of the system resources.
- **Service level C**. Simulation requests will be executed in parallel by using remaining system resources.

These three levels are assured to the user but if there are no simulation requests with higher priority the system automatically moves the request into the higher level. Every artificial neural network paradigm y is offered for the price of q for single use or a price Q to use a paradigm for m months. Only evaluation and eventually training of a neural network will be charged, the creation of a neural network should be free of paradigm vendor fees. Users are able to choose between seven different pricing models - *Standard, Premium* and *Minimal* are offered on a pay-per-use base whereas *Flat rate* and *Local execution* have an assessment period of one month.

- **Premium.** Users pay a price of p_A for every processing cycle x and q for each selected paradigm y. The guaranteed service level is A.
- **Standard.** Users pay a price of p_B for every processing cycle x and q for each selected paradigm y. The guaranteed service level is B.
- **Minimal.** Users pay a price of p_C for every processing cycle x and q for each selected paradigm y. The service level is C but will often be B.
- **Flat rate**. Users are charged with the pice P for the selected SLA for m months with a limit of x computation cycles per month. If users exceed this limit, they will be downgraded to the lower service levels during the rest of the month. Only if they choosed a package with service level C and they exceed the network traffic limit tthey are asked to use their credits for the following month or to buy a new package.
- **Local execution.** For experienced users it is also possible to download paradigm/s to his own machine (notebook, workstation or server) so he has to pay only for these paradigm/s plus a small manipulation fee to the Cloud provider. Each paradigm is secured by a license key which expires at the end of m months, so the paradigm subscription has to be renewed if the consumer still wants to use them.
- **Negotiated roles.** Large customers having special requirements are given the opportunity to agree to special packages which can then be reused for similar customers.
- **Dynamic negotiation.** If customers want to use N2Sky extensively it is possible to dynamically negotiate the terms of use with both the Cloud provider and the paradigm vendor where the negotiations should be moderated by the system.

As a variant of the pricing models *Standard* and *Minimal* limits for the Cloud provider prices p_B and p_C could be set by the customer within a predetermined range similar to bids for spot prices in the context of Amazon EC2 [29] spot instances. In this model, spot rates are constantly changing depending on system load. Simulation tasks are executed only if the spot price is not higher than a fixed limit.

In addition to pricing models in the context of using neural network paradigms the pricing model for hosting arbitrary software components (see Subsection 3.2.9) can be chosen similar to the *Amazon EC2 instance purchasing options* (*On-Demand Instances*, *Reserved Instances* and *Spot Instances*).

These pricing models are not only applicable to CPU processing cycles but also to other harware resources like storage, infrastructure and special instrument where x can be seen for example as disk space in megabyte, network traffic in kilobyte or square meters of pages printed wich a special plotter.

3.2.4 Controlling and Accounting

With the term *Controlling* we mean costing services provided by N2Sky mainly based on three components: *Simulation Management, Business Administration and Monitoring*. A common controlling component is SAP CO providing a large variety of costing methods.

Accounting manages billing and tax handling and can be performed using SAP FI. As alternative to costly SAP components there are a variety of freely available ERP (Enterprise Resource Planning) systems like OpenERP [30], SQL-Ledger [31], ADempiere [32], LedgerSMB [33] or Compiere [34]. One of the most interesting open source ERP systems is OpenBravo [35] because its GUI is fully web-based and its Java web application runs on Tomcat. OpenBravo is partly based on Compiere and works with PostgreSQL or Oracle.

To minimize accounting costs a payment method has to be chosen already during user registration where end users have to transfer money to their personal credit accounts, preferably by bank transfer, *PayPal* or *Google Checkout.* In the future also secure mobile credit card payment methods are being considered as described in the mobile gSET [36] research project.

Figure 3.2 depicts relations between stakeholders, resources and organizations. A stakeholder is assigned to a *Cost Bearer Unit* of a company or an other organization which receives the bill for every N2Sky service used by employees of that unit.

3.2.5 User and Role Management

3.2.5.1 Stakeholder

In the context of Virtual Organizations *Stakeholders* consume and produce resources offered by a VO. The IEEE Standard 1471-2000 defines a stakeholder as user of the system, developer and provider of the system's technology, maintainer of the system and one responsible by the acquisition of the system [6]. Figure 3.3 depicts the N2Sky stakeholder hierarchy derived from the stakeholder hierarchy presented in [37]. The four yellow boxes represent instantiable stakeholder classes.

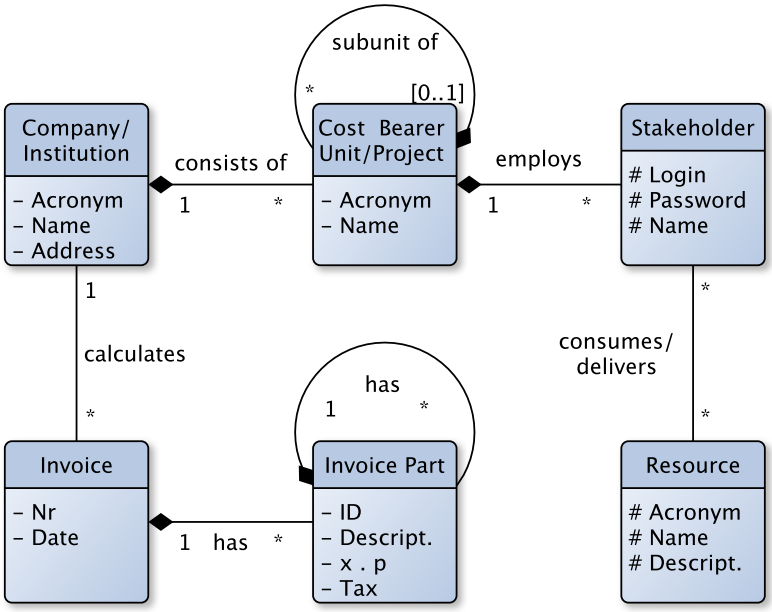

Figure 3.2: Invoicing of services consumed by employees working on a project.

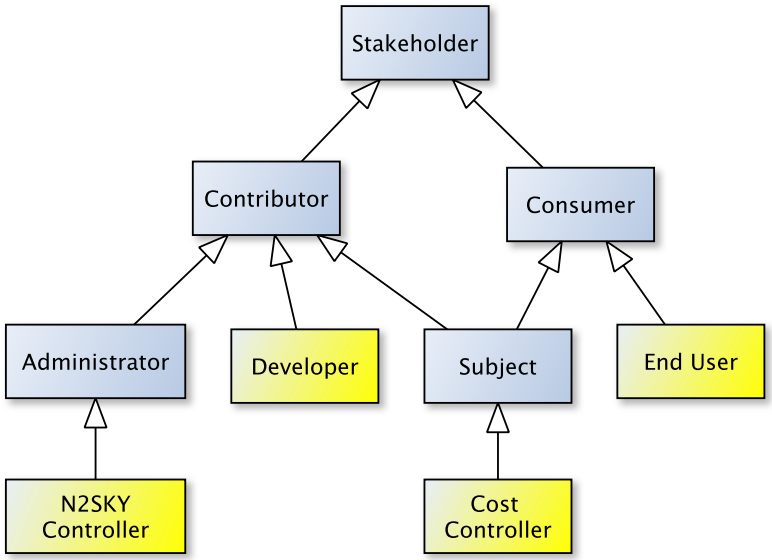

Figure 3.3: The stakeholder hierarchy containing abstract and concrete (yellow) roles.

Each user (i.e. stakeholder) can own an arbitrary number of roles for a particular subtree of the resource tree structured hierarchically (see Figure 3.5). There are four different basic roles: (see Table 3.2):

- **N2Sky Controller.** A *N2Sky Controller* is able to add (+) and remove (-) any role to any user over a graphical user and role management interface.
- **Developer.** The *Developer* of neural network resources has all permissions within his unit except of the manipulation of Cost Controller roles.
- **Cost Controller.** A *Cost Controller* is responsible for expenses of a particular cost bearer unit and has all permissions within this unit.
- **End User.** A *End User* is able to consume services up to the defined budget limit per month. Budget limit operations have to be approved from the Cost Controller responsible for this cost bearer unit.

Permission to	N2Sky Contr. (NC)	Developer	Cost Contr. (CC)	End User
+/- NC Role	x	-	-	-
Reset Password	x	-	within Unit	-
+/- CC Role	x	-	within Unit	-
+/- Devel. Role	x	within Unit	within Unit	-
+/- End User R.	x	within Unit	within Unit	withinUnit
Set Budget Limit	CC approvement	CC appr.	x	CC appr.

Table 3.2: System-wide and unit-specific user permissions

3.2.5.2 Subject

Khalil [37] divide stakeholders into four categories (*Consumer*, *Contributor*, *Developer* and *Administrator*) according to the IEEE Standard 1471-2000 [6] and introduces a new stakeholder type: the *Subject*. A *Subject* is defined as "a component of a Virtual Organization, which can consume the resources offered by a Virtual Organization and also can act like a resource to be consumed in the Virtual Organization environment." [37] and redefines a Virtual Organization as "a set of cooperating building blocks, called *Subjects*".

In the view of above definition a *Subject* has two main characteristics:

1. *Subject* can be represented as a subclass of both *Contributor* and *Consumer or*
2. *Subject* can be represented as a subclass of *Expertise* which in turn is a logical resource.

The first case is shown in the UML class diagram in Figure **??** where *Subject* is a subclass of both *Contributor* and *Consumer*. For example a researcher runs simulations as a consumer of N2Sky, on the other hand he offers a new paradigm which he has even developed. In this case there are two billing options: If the developed paradigm is offered

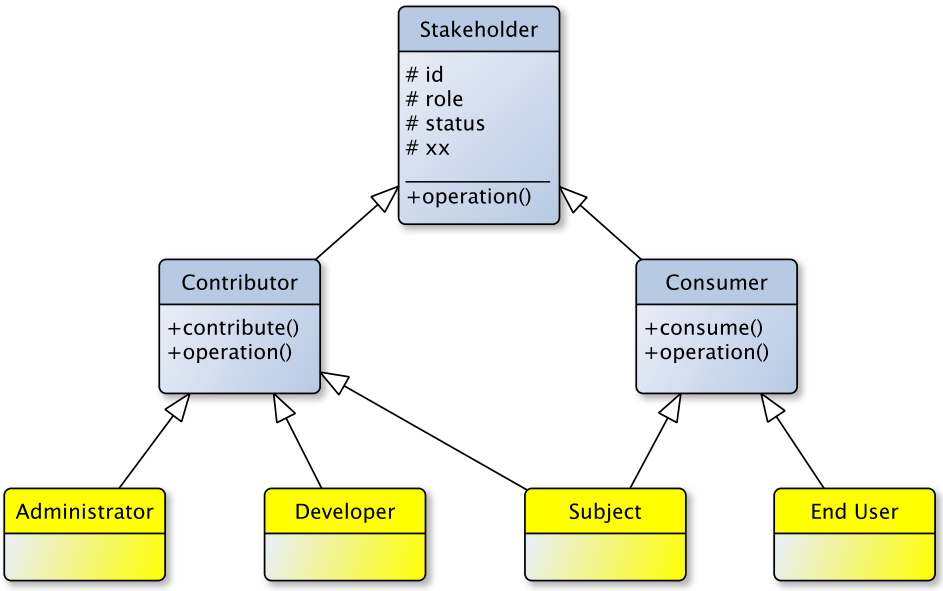

Figure 3.4: *Subject* is a subclass of both *Contributor* and *Consumer*.[3]

free of charge, the researcher will get a reward from the Cloud provider otherwise he earns money depending on how often his paradigm is demanded.

The second case is shown in the class diagram in Figure 3.5 where *Subject* is a subclass of *Expertise* which in turn is a logical resource within the resource hierarchy. In addition to the resource hierarchy presented in [37] *Formal Knowledge* and *Expertise* are seen as separate subclasses of *Logical Resource*. Formal knowledge is stored in knowledge bases as well-structured facts and can be retrieved over various deduction mechanisms by combining single facts. *Subject* as a subclass of *Expertise* represents a human expert offering personal advisory services. Yellow boxes represent the basic billing unit for each resource where the aggregation relationship indicates that each resource can be divided into uniform billable units. Within the subtree of physical resources *network resources* are extracted from *infrastructure resources* because we think that network resources are as important as computation and storage. In return, infrastructure resources are specified as standard peripheral infrastructure resources (e.g. printer, scanner, camera).

3.2.6 Access Control

There are a variety of security aspects within a SOA. We concentrate on authentication, authorization and *Public Key Infrastructure (PKI)*. To establish business models described in Subsection3.2.3, it is absolutely necessary to secure services provided by N2Sky. Allowing stakeholders to gain privileges for an entire collection of resources, every resource is structured within trees (see Figure 3.5), in other words, the stakeholder owns a single resource or even a complete resource subtree.

[3]Source: Own illustration on the basis of [37, p. 21].

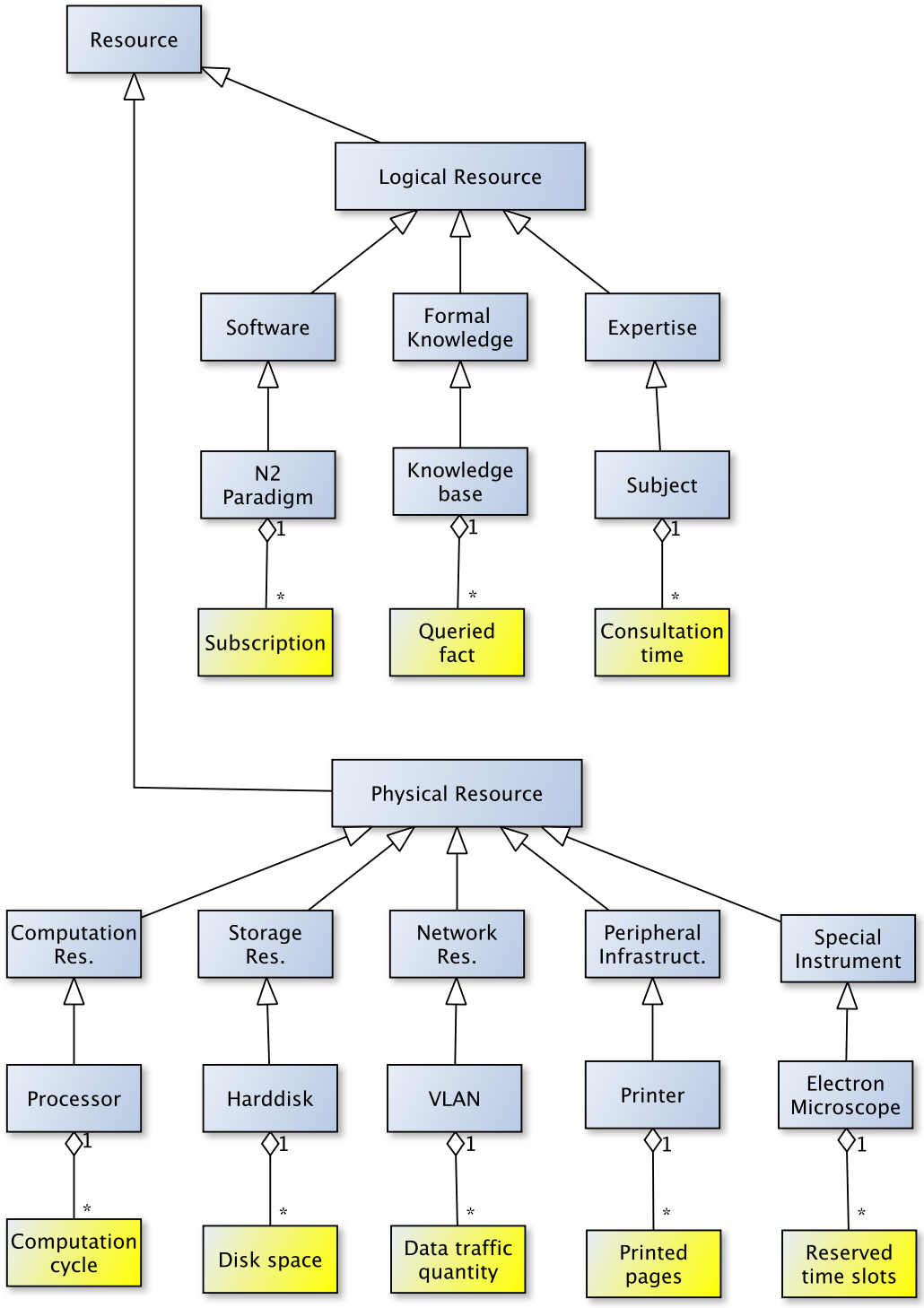

Figure 3.5: Resource hierarchy with units of account (yellow boxes).

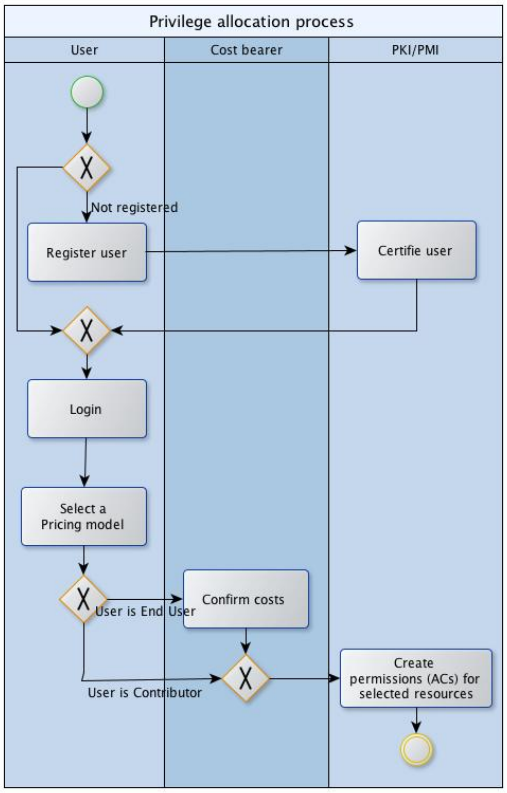

Figure 3.6: The *Privilege allocation process.*

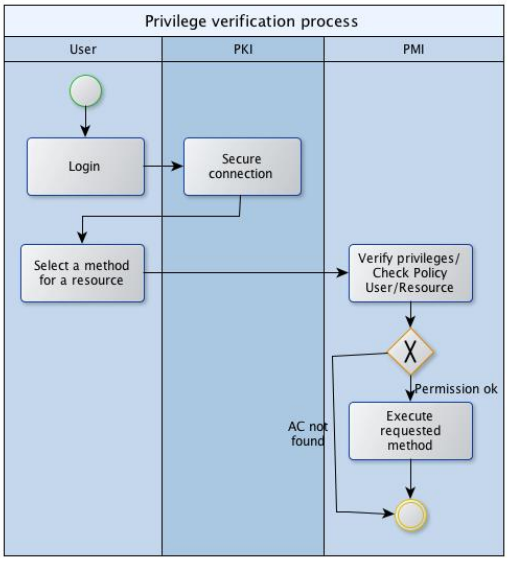

Figure 3.7: The *Privilege verification process.*

The BPMN model in Figure 3.6 depicts the *Privilege allocation process*. During user registration the PKI generates a keypair for a user certificate. After login the user selects a pricing model as described in Subsection 3.2.3. If the user is an *End User* and has

no *Cost Controller* role (i.e. the employing organization bears all cost arising in this unit), an approvement of this manager is necessary to allocate privileges for requested resources.

The BPMN model in Figure 3.7 shows the *Privilege verification process*. The PKI checks the user certificate and establishs an encrypted connection. After selecting a particular service, method and resource the *Privilege Management Infrastructure* (PMI) checks needed user policies and decides if the requested operation will be permitted or denied.

3.2.7 Workflow System

There is a large number of scientific workflow systems, most of them are designed to run in Grid or Cloud infrastructures.

Kepler. The scientific workflow system Kepler [38] allows combination of activities to a single process. Osterweil calls Kepler as "perhaps the most advanced scientific workflow system project" [39] - among Chimera, Taverna, JOpera and Teuta. Kepler is based on **Ptolemy II** and integrates a broad range of tools to support specification, visualization and execution of scientific workflows. During process execution, input and output data, user data and time stamps will be logged. Regarding logging of intermediate steps you have to consider an important aspect: If an activity returns a very large result, it should be excluded from step-logging for reasons of performance.

Taverna. The scientific workflow system Taverna [40] is mainly used for bioinformatics applications. It contains two different types of data provenance: on one hand, a simple model that tracks data flows and data dependencies between activities and on the other hand a more advanced model for lightweight annotations to the workflow steps. By using semantic annotations the context of activities is described even better

Chimera. Chimera [41] is able to find or create workflows of Grid services within a scientific environment. The **Virtual Data Language (VDL)** of Chimera allows for the description of abstract workflows while the included **Pegasus Planner** maps from abstract to concrete workflows including data movement jobs for executing workflows in a Grid environment. Scientific processes are visualized by **Data flow graphs (DFGs).**

JOpera. JOpera is an Eclipse-based workflow system for modeling and executing heterogeneous Grid workflows, It is developed at ETH Zürich and can be used to compose processes consisting of RESTful Web Services to meet requirements like dynamic late binding, content-type negotiation, dynamic typing, and state inspection [42].

Teuta. Teuta. [43] is a graphical editor for composing uml-based scientific Grid workflows and is integrated with the **Askalon Grid Environment (AGE)** [44]. In the context of AGE, Teuta is used for composition, submission, controlling and monitoring of scientific Grid and Cloud workflows. The included **Domain Specific Language (DSL)** for specifying Grid workflows is based on UML 2.0.

3.2.8 Knowledge Management

The *Knowledge Management* component should provide ontologies helping to solve various problems, especially supporting the *Query Interface* described in Subsection 3.4.1. RDF and OWL are possible description languages for ontologies and Sesame [45] is a framework for storing and retrieving RDF data as language for describing ontologies.

3.2.9 Component Hosting Platform

The idea of a *Component Hosting Platform (CHP)* is to deploy and run arbitrary software components on a Cloud computing environment like a "Google App Engine for neural network components". This should enable N2Sky stakeholders to outsource any software components, in particular those using neural network services. This results in the following functional and non-functional requirements:

Functional requirements

- Component interfaces via Web Services (SOAP or REST).
- Provision of different runtime environments: Java, .net, Ruby, ...
- Provision of different application servers: Apache, Tomcat, JBoss, Microsoft IIS, ...
- Provision of different DBMS: MySQL, PostgreSQL, Oracle, MS SQL Server, ...
- Ability to manage databases within the customer's environment.

Non-functional requirements

- Provision of a higher computing capacity at peak load (elasticity).

3.2.10 Annotation Service

We believe that on a comprehensive graphical interface like *N2Sky's Web Portal* there should be not only room for official scientific facts, but also for informal, personal information. For this purpose we will develop the hideable *Annotation Service.* If this interlayer appears a note for each described object will be added. Descriptions can contain both formal and informal information each labeled with a different color. Notes can be arranged directly at the object or in a separate column on the right side of the screen depending on user settings. Every user is able to add comments to any object in order to facilitate discussions. Scientists should be encouraged to add all those details not usually included in academic work, especially experiments that have failed and pitfalls to be avoided.

The *Annotation Service* should also include Web 2.0 features like user ratings for neural network paradigms and other objects, instant messaging with other users or queries about individuals (other blog entries, ratings, other offered neural network paradigms). For this purpose, an integration of Twitter or Xing is conceivable.

3.3 PaaS: Neural Network Layer

3.3.1 Simulation Services

Paradigm Simulation Services provide standardized neural network object functionality (`train()`, `evaluate()`, `checkStatus()`, `getResult()`, ...). Types and operations are described in detail in Subsection 4.3.1.

3.3.2 Simulation Management

The purpose of the *Simulation management* component is to manage neural network simulation tasks by receiving commands initiated from actions on the web portal or smart phone app. On the other side this component forwards commands and data to paradigm-specific neural network simulation services over a standardized interface that is mandatory for all simulation services.

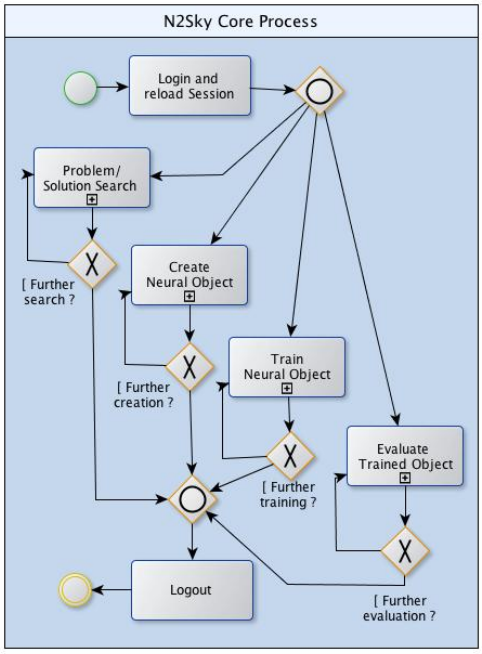

Figure 3.8: The N2Sky core process.

The core process of the *Simulation management* component is depicted in Figure 3.8 by using the functionality of simulation services which is described in Subsection 3.3.1.

After user authentication has been performed successfully, your previous session will be reloaded. It allows users to find all their subscribed, created, trained or evaluated neural network paradigms and objects.

The four subprocesses of the core process are the *Problem/Solution Search* described in Subsection 3.4.1 (Query Interface) as well as creation, training and evaluation of neural network objects.

3.3.2.1 Create Neural Object

The BPMN diagram in Figure 3.9 depicts the creation process of a neural network object either by duplicating an existing neural object or by creating a new one. In both cases the user has to follow these three steps whereas all default values are copies of defined values of the original neural object.

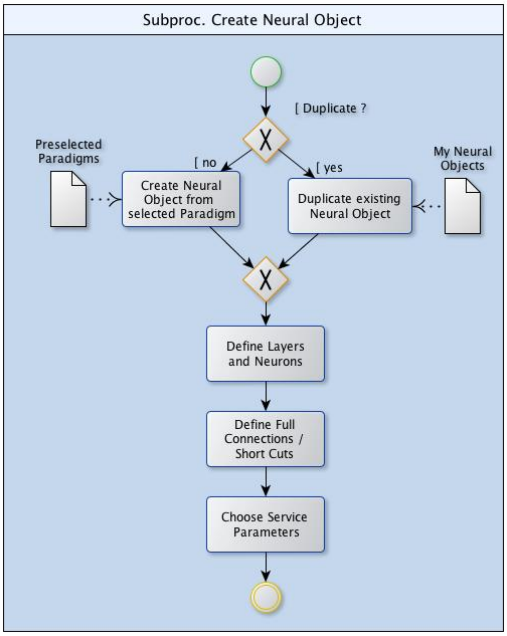

Figure 3.9: The subprocess *Create Neural Object*.

1. **Layer definition.** The first step in defining a neural object is to define its layers by defining the number of input and output neurons as well as the number of hidden layers and the number of hidden neurons of each hidden layer. The maximum and minimum possible number of neurons and layers are displayed just to the left of the input field.

2. **Layer connections.** On top of the screen user inputs regarding the neural network (input and output neurons, hidden layers and neurons) are displayed graphically in a matrix form. The standard value for the connection model is *Full-Connections*. If the user does not want to have full connections between particular layers he has to deselect these layers and can create his own shortcuts by filling out appropriate input fields.

3. **Parameter definition.** This page is generated entirely dynamically by using the selected paradigm's service description including default values of every input field.

3.3.2.2 Train

The BPMN diagram in Figure 3.10 depicts activities within the training subprocess which consists of the following five activities:

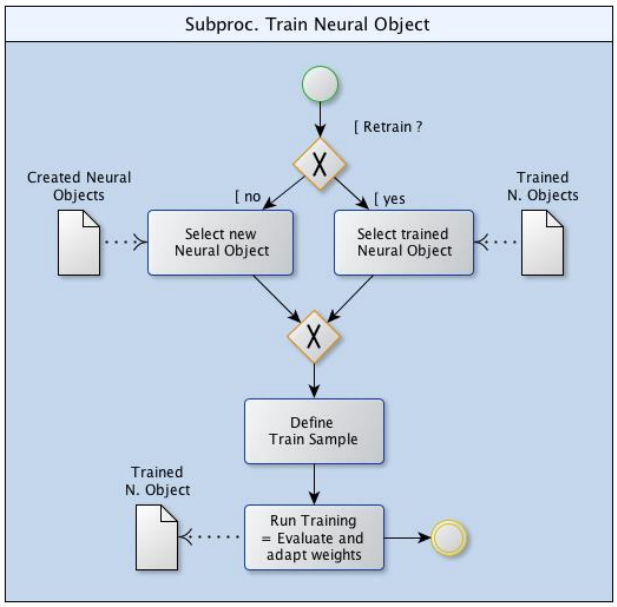

Figure 3.10: The subprocess *Train Neural Object*..

1. **Select object.** Select a recently created or an already trained (retrain case) neural object.
2. **Train sample definition.** There are two ways to set input samples: Type or copy input and output values into the provided text area or upload a sample data text file. In both cases every values has to be separated by using semicolons . The number of values per sample has to be exactly the same as defined during creation (i.e. the number of input and output neurons).
3. **Training and status info.** After pushing the `Train` button the `train` operation of the *Simulation service* is called in an asynchronous way, i.e. users don't need to wait until training has finished, rather they are able to run other tasks at the same time. Next to each object its completion status is displayed based on the following traffic light system:
   (a) **green progress bar:** Shows computation status of a training.
   (b) **green or paradigm icon:** Training completed successfully,
   (c) **red:** Training aborted with error(s).

4. **Training results.** Trained input and output values are shown. If this option is selected a training curve is generated and displayed as an interactive diagram.
5. **Retrain.** If deltas between sample results and calculated results are not sufficiently low, the user is able to restart training with further training samples.

3.3.2.3 Evaluate

The BPMN diagram in Figure 3.11 shows activities within the evaluation subprocess which consists of the following five activities:

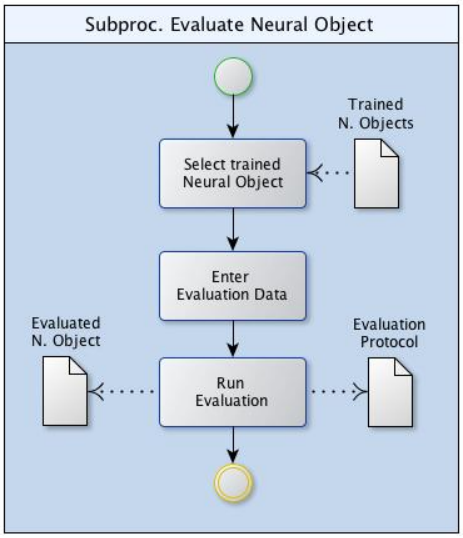

Figure 3.11: The subprocess *Evaluate Neural Object*.

1. **Check access rights.** The *Access control* component checks if the user is entitled to evaluate a particular neural object. Permissions are based on the pricing model the user has chosen.
2. **Insert evaluation values.** There are two ways to insert input values: Type or copy input values into the provided text area or upload an evaluation data text file. In both cases all values have to be separated using semicolons. The number of values per line has to be exactly the same as defined during creation (i.e. the number of input neurons).
3. **Evaluation and status info.** After pushing the `Evaluate` button the `evaluate` operation of the *Simulation service* is called in an asynchronous manner. Next to each object its completion status is displayed based on the following traffic light system:
   (a) **green progress bar:** Shows its computation status
   (b) **green or paradigm icon:** Evaluation completed successfully
   (c) **red:** Evaluation aborted with error(s).

4. **Evaluation protocol and result.** After evaluation was completed, the *Accounting and Controlling* component is supplied with usage data for later billing. Only then evaluation inputs and results are shown on the screen.

5. **Further evaluation.** If further evaluation of the same neural object is desired users are able to input further evaluation data.

3.3.3 Business Administration

3.3.3.1 Use Cases

The *Business Administration Service* provides operations for managing N2Sky's business workflow depicted in the following Use Case diagrams. It is intended to develop a component inspired by Apple's *iTunes Connect*, a suite of web-based tools to manage publishing and sales processes of both Mac [46] and iOS [47] apps in the Mac/iOS app store. The diagram in Figure 3.12 shows relations between use cases during integration of resources and users. *End users* and contributors (both simply called *User*s) have to register before they are able to use and contribute to N2Sky. To manage commercial or institutional users, every user is assigned to a unit or a project which in turn belongs to a company that acts as a legal business partner. To utilize services immediately, users have to select a payment method and have to credit their accounts by making prepayments. Software and hardware providers are able to integrate neural network paradigms and hardware and have to define terms of use or select one of the proposed terms. The entry point for *End users* is the *Query interface* where they are able to search for problems or paradigms (as problem solution methods).

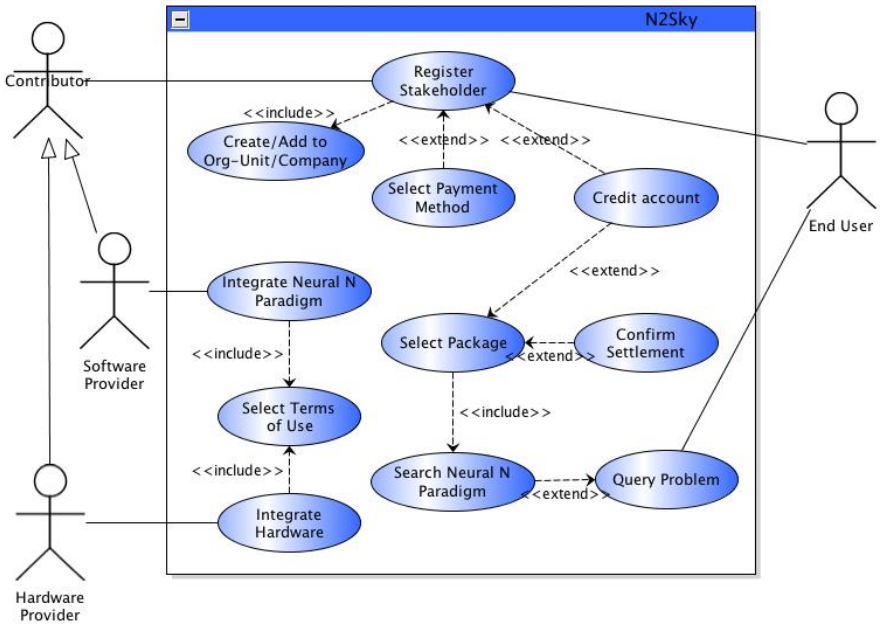

Figure 3.12: Integration of users, software and hardware.

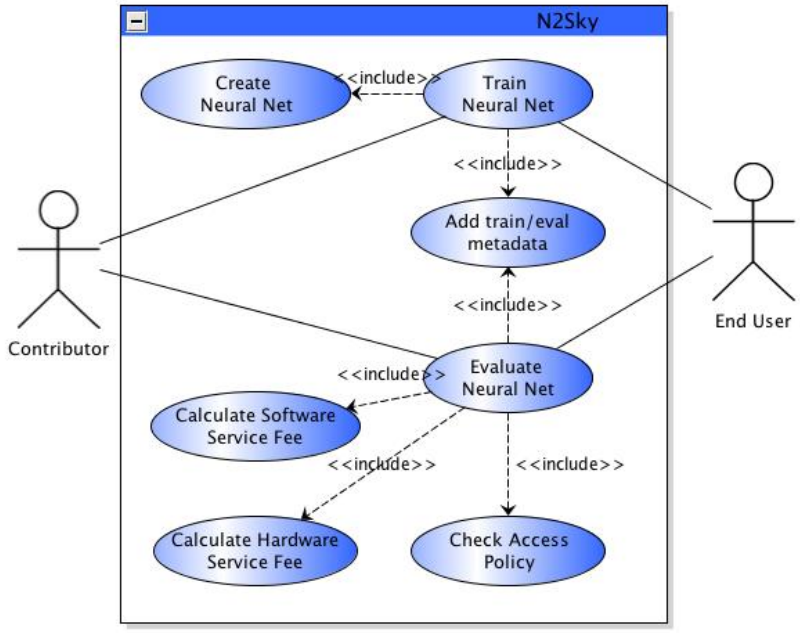

Figure 3.13: Training and evaluation of neural networks.

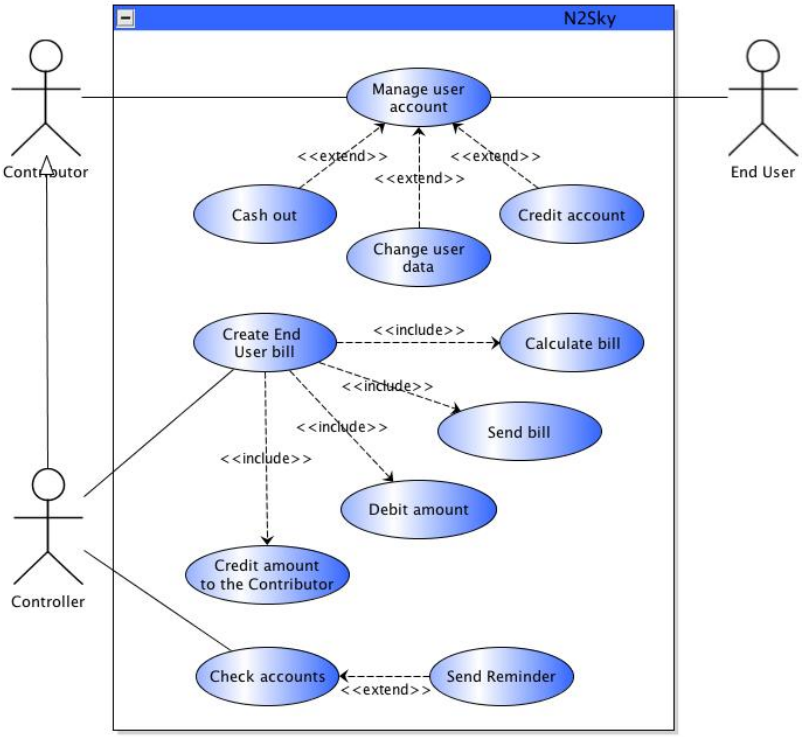

Figure 3.14: Financial controlling Use cases.

The Use case diagram in Figure 3.13 shows actions during execution of neural network sessions. Software and hardware service fees are calculated only during evaluation and eventually during training of neural network objects.

The Use Case diagram in Figure 3.14 depicts relations between use cases in the financial controlling area. The administrators own account includes not only changing of user data but also transfer of money from and to his own account. A *Controller* is derived from the *Contributor* actor and is responsable for financial controlling of the whole N2Sky system. Creating *End user* bills contains calculation and clearing of the invoice amount. Furthermore a *Controller* is also responsible for checking all user accounts regularly and for sending them reminders if there is not enough money on their accounts. Such a business workflow makes sense only if adequate security mechanisms are implemented, particularly access control and secure data transfer.

3.3.3.2 Domain Model

The UML class diagram depicted in Figure 3.15 shows structure and interconnections of classes belonging to two different domains: the *Neural network domain* and the *Business domain*.

3.3.3.2.1 The Neural Network Domain The *Neural Network domain* consists of *Paradigm, Creation* and *Simulation*, whereas *Training* and *Evaluation* are subclasses of *Simulation*. *Training* always belongs to a *Creation* and *Evaluation* always belongs to a *Training* object and does not exist without it.

- **Paradigm.** A neural network paradigm is a subclass of *Resource* described in Subsection 3.2.5. The resource hierarchy in Figure 3.5 depicts its superclasses: A *Paradigm* is a *Software, Logical Resource* and finally, a *Resource*. The unique paradigm acronym consists of capitalized characters and numbers whereas the first four characters always define the paradigm group providing the ability to subscribe for more than one paradigm at once. For example, `BPROTUTO` is the acronym for the *Tutorial Backpropagation* paradigm, a demo paradigm service especially to demonstrate the interface mechanism of N2Sky paradigm services. `BPRO` is the paradigm group *Backpropagation*. An acronym can be seen as a "public" primary key so that it should be relatively simple to change it whereas the real primary key (numerical and automatically generated) will be untouched. Every paradigm has exactly one neural network service description (`NNServiceDescription`). The XML Schema definition (`n2sky-description.xsd`) is listed in Appendix A.1.
- **Creation.** A *Creation* is an instantiation of a neural network paradigm and results to a neural object consisting of defined parameter values and possibly of input sample data. Every *Creation* results in exactly one neural network service definition (`NNServiceDefinition`). The XML Schema (`n2sky-definition.xsd`) is listed in Appendix A.2. A neural network object can be used on two ways:
  - **Creation-based** on free use of a particular paradigm.

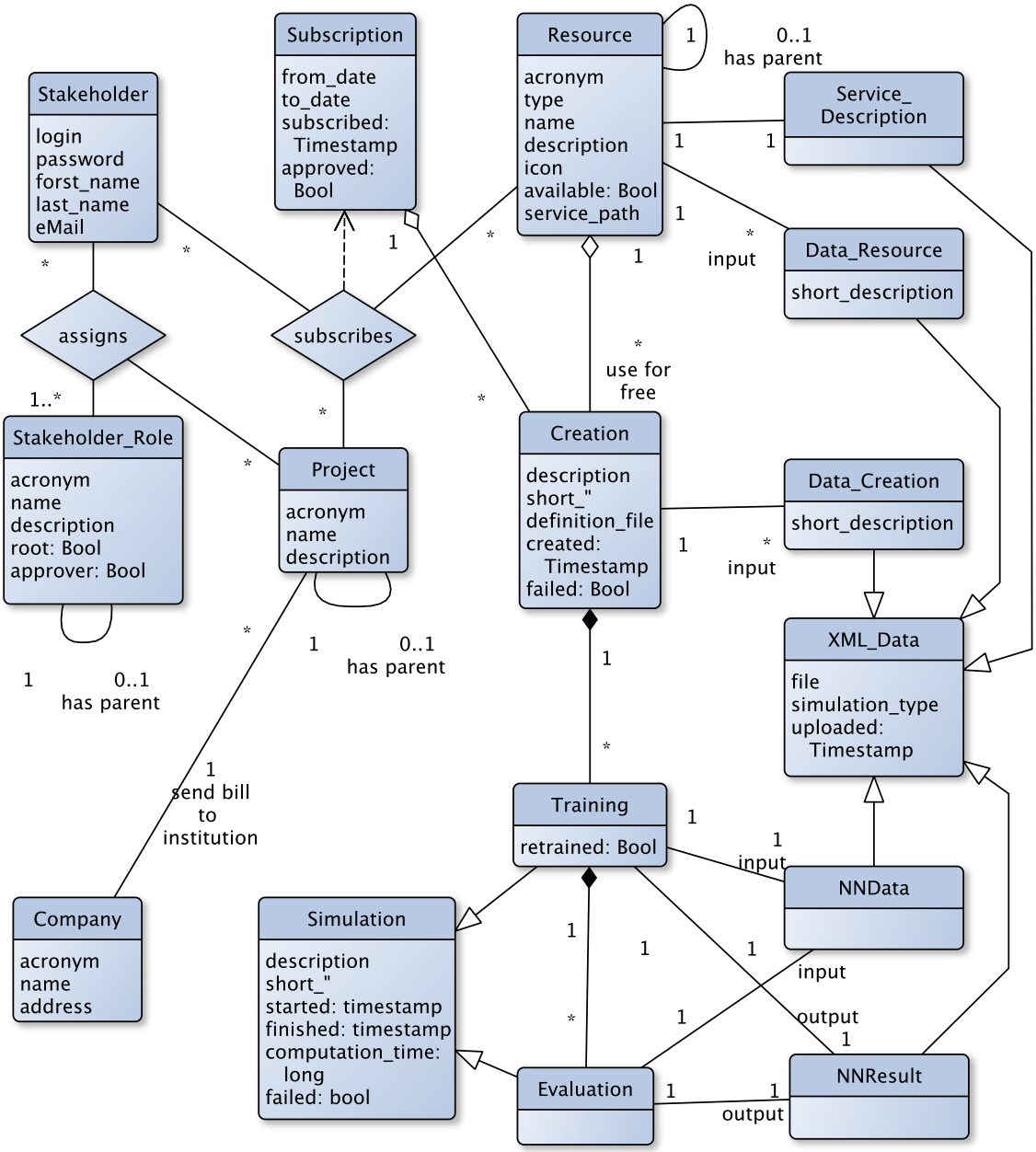

Figure 3.15: N2Sky's domain model as UML class diagram.

- **Subscription-based** creation of a neural object if the paradigm provider decided to earn money for provided services based on monthly subscriptions.

- **Simulation.** The two neural network simulation methods are *Training* and *Evaluation*. Since these methods are time-consuming depending on sample data size and error thresholds. *Starting* and *finishing time* as well as *computation time in milliseconds* are saved. For applying more advanced business models, type and number of CPU cores as well as the number of CPU micro instructions can be added to these classes. Every *Simulation* consumes exactly one *neural network data document* (XSD listed in A.3) which consists of an arbitrary number of input/output data samples (for training) or onyl input samples (for evaluation) and results in exactly one *neural network result* (`NNResult - n2sky-result.xsd`) listed in Appendix A.4.

3.3.3.2.2 The Business Domain The *Business domain* consists of *Stakeholder, Role, Subscription* and *Project*.

- **Stakeholder and Role.** Every *Stakeholder* owns at least one *Role* (in most cases an *End user* role). A stakeholder's *Role assignment* consists of one assigned role and one project the stakeholder works for. So if a stakeholder woks for more than one project and owns different roles within these projects, this case can be configured in N2Sky by adding several *Role assignment*s. Every role except the root role has a parent role so that a *Role hierarchy* described in Subsection 3.2.5 can be customized.

- **Subscription.** If a resource is not for free you can make a *Subscription* for a particular time period. To subscribe for a resource it is also mandatory to select a *Project*. For billing purpose it is important to know for which *Project* the *Resource* is used. This project has to be part of the list of *Stakeholder-Role-Project* assignments of the particular stakeholder.

- **Project.** Our research is in the field of virtual organizations (VOs), so we use the term *Project* instead of *Unit* (i.e. department) of a company. Every *Project* has a unique acronym (capitalized characters and numbers) and has exactly one parent project for making project hierarchies possible. Furthermore, every *Project* belongs to a *Company* or institution to enable a correct fiscal billing process.

3.3.4 Hosted Components

Hosted components could be arbitrary software components managed by the *Component Hosting Platform* described in Subsection 3.2.9.

3.4 SaaS: Service Layer

3.4.1 Query Interface

The *Query Interface* (based on the *Problem and Solution Search* depicted in Figure 3.16) should provide a search engine interface like Google's search bar to search either for optimization problems or solution methods for such problems on predefined search criteria. In the context of neural networks this solutions are called neural network paradigms (e.g. *Backpropagation*, *Quickpropagation*, *Kohonen net*, *Jordan net*, *Self organizing maps*). Expert's knowledge can be retrieved to solve this *Paradigm matching problem* by using the *Knowledge Management* component (see Subsection 3.2.8). Combining one or more paradigms with a pricing model can lead to a business case as described in Subsection 3.2.3.

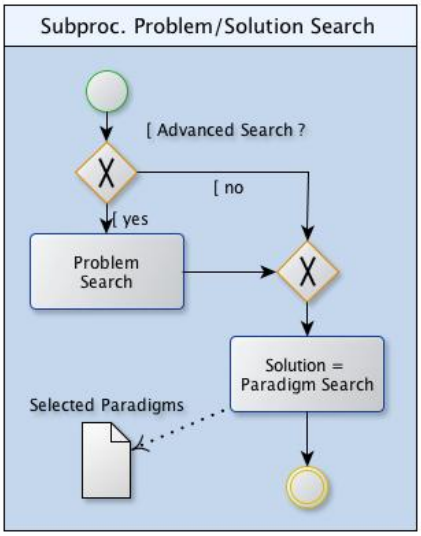

Figure 3.16: The *Neural Prroblem/Solution search* subprocess.

3.4.2 Web Portal

The Web Portal is the central entry point both for service consumers and providers. As an extended version of the *Smartphone app*, it is directly interconnected with *Simulation Management* and *Business Administration.*

3.4.3 Smartphone App

The *Smartphone App* is the central entry point primarily for service consumers and provides a graphical user interface (GUI) for the *Simulation Core Process* described in Subsection 3.3.2 . So we see this app as the *Neural network simulation portal* mandatory for N2Sky. The extended *Web portal* described above provides additional business

administration tasks both for service consumers and providers. Multiple versions of the app (iOS, Android) are automatically generated from a mobile web portal of the same functionality

3.4.4 Hosted UIs

Hosted User Interfaces are UIs for *Hosted Components* described in Subsection 3.3.4.

Chapter 4

Component Interfaces

In this chapter interfaces of components discussed in previous chapter will be specified. A comparison between some of these interfaces and corresponding RAVO interfaces prepared in common with Wajeeha Khalil has already been published in [14] (Appendix A). According to N2Sky's architecture presented in Section 2.2, every component will be provided as a service (over protocols, Web Services or APIs). The component interface design of N2Sky is driven by following guiding principles:

- **Acceptance.** To be accepted by the user the system has to provide a flexible and intuitive user interface with all necessary resources in an easy-to-use way.
- **Simplicity.** The system has to supply functions for manipulating neural networks in a simple and (more important) natural manner. This can be achieved by an adequate (i.e. a natural) representation of neural objects in the provided environment of the system A neural network object has to react and to be administrated as any other conventional data object during its creation, update and deletion.
- **Originality.** Even though a neural object has to be simple, it has not to loose its originality. A common framework always runs the risk to modify the original property structure of the individual object. This leads to the situation that either objects of different types loose their distinguishable properties or loose some parts of their functionality. Therefore a suitable neural network simulation framework has to pay attention to the properties and to the functional power of neural objects and should give the user an undistorted view on it.
- **Homogeneity.** Neural objects have to be considered as conventional objects, which can be stored in any data management environment, either in database systems or in distributed Cloud data stores. From the logical point of view a neural object is a complex data value which can be stored as any other data object.
- **System extensibility.** N2Sky offers an easy-to-use interface for neural network researchers to extend the set of neural paradigms provided as services. This can be done by uploading and integrating additional paradigms. After successfully passing the paradigm publishing process it will be made available to any other user.

Following these principles components in the SaaS layer are designed to adapt itself to various platforms (conventional or mobile web browsers running on workstations, PCs, Macs, tablets or smart phones) via HTML5, CSS3 and JavaScript. As we use The-M-Project [48], a MVC framework for mobile (web) applications based on jQuery Mobile, it is possible to create native iPhone, iPad or Android apps based on our mobile web application with minimal additional effort.

The rest of this chapter presents interface specifications for every component described in the previous chapter. Each interface specification table is structured as follows:

- **Configuration.** It is a design goal that almost every component should be configurable in a simple and flexible way. In most cases configuration is implemented over configuration files.
- **Inbound Interface**. This interface is mandatory and is used to control the particular component itself.
- **Outbound Interface.** This interface is optional and is used to control other components in lower layers, e.g. the *Smartphone app* calls services of the *Simulation management* component via RESTful Web Services. Outbound interfaces are not explicitely annotated in the following tables because they are equal to the inbound interfaces of components listed in line "*Used components*".
- **Interface type.** The suggested interface technology, e.g. Web Services (WS), Application Programming Interfaces (APIs) or special protocols.
- **Parameter name and type.** Specified parameter names (input params: IN, output params: OUT) and data types if the interface technology supports data types.
- **Mandatory (mand.).** Indicates if configuration, particular configuration parameters, interfaces, operations or used components are needed in every case.
- **Synchronous/asynchronous.** Indicates if any underlying operation is time-consuming (asynchr.) or not (synchr.). For example a `train` method of the neural network simulation service calls the `train` method or several methods if training is distributed over multiple cores, processors or Cloud nodes of a paradigm service. Duration time depends on paradigm complexity, training data size and further parameters and can take several seconds, hours or even days. Synchron methods on the other hand should execute within a few milliseconds. It is important to this definition of asynchron to distinguish it from asynchronous service or method calls like *Asynchronous JavaScript and XML (AJAX)* where the next program step is executed just after calling the Web Service without weiting for a response. If the Web Service finished execution it calls the service consumer's callback function to continue executing statements that depend on this service result.
- **Integration Phases.** Regarding to our integration plan there are three types of components:
  - **Mandatory (mand.).** These core components already existing likewise in former versions of N2Sky (i.e. N2Grid [20] and N2Cloud [21]).

– **Integration Phase 1 (P1).** Additional components partially integrated described in Figure ??.

– **Integration Phase 2 (P2).** Even more components which are not integrated so far. These mostly in this paper first presented components are depicted in Figure 2.2.

4.1 IaaS: Infrastructure Enabler

4.1.1 Component Archive

The interface specification of the *Component Archive Service* as shown in Table 4.1 contains methods replicating components in a dynamic way, i.e. they are retrieved from the archive only at that moment on which they are needed (before the `train` method is called). In this context components and hosts are the key resources. In a JVM-based container a component is a `.war` file containing the Java web application and a host is a Java web application container like Tomcat running on an instance on a Cloud node.

Existing instances of the *Component Archive Service* are:

- N2Sky Component Archive
- N2Grid Component Archive

Component Archive (mand.)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			*
	registryURL	URL	*
Inbound interface		WS, API	*
archiveNewComponent()		asynchr.	*
...			
copyComponentToHost()		asynchr.	*
...			
getAllArchivedComponents()		synchr	*
...			
getArchivedComponentsOnHost()		synchr.	*
...			
hasReplica()		synchr.	*
OUT:	hasReplica	boolean	*
deleteComponentFromHost()		asynchr.	*
...			
Used components:	Registry		*
	Cloud Infrastructure		*

Table 4.1: Interface specification of the *Component Archive*.

4.1.2 Data Archive

The interface specification of the *Data Archive* is described in Table 4.2. The *N2Sky Data Archive* contains only a subset of the functionality of OGSA-DAI [25], namely *DRES, Data sink service* and *Data source service.*

Existing instances of the *Data Archive:*

- N2Sky Data Archive

Data Archive (mand.)		
Interface, service, description	type	mand.
Configuration		
Inbound interface	WS, API	*
Data request execution service (DRES)	synchr.	*
Is used to submit workflows, create sessions and get the status of synchronous requests.		
Data resource information service (DRIS)	synchr.	*
Is used to query information about a stored resource.		
Data sink service	synchr./asynchr.	*
Is used to push data to data sinks.		
Data source service	synchr./asynchr.	*
Is used to pull data from data sources.		
Session management service	synchr.	*
Is used to manage the lifetime of sessions.		
Request management service	synchr.	*
Is used to query request execution status subsequently of asynchronous requests.		
Used components:	Filesystem	*
	DBMS	

Table 4.2: Interface specification of the *Data Archive. according to OGSA-DAI*

Ad-hoc Infrastructure (P2)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			*
	registryURL	URL	*
Inbound interface		WS, API	*
registerInfrastructureComponent()		synchr.	*
...			
removeInfrastructureComponent()		synchr.	*
...			
getPricingModel()		synchr	*
...			
Used components:	Registry		*
	Infrastructure components		*

Table 4.3: Interface specification of the *Ad-hoc Infrastructure component.*

4.1.3 Ad-hoc Infrastructure

The interface of the *Ad-hoc Infrastructure* component described in Subsection 3.1.3 is shown in Table 4.3. Currently there is still no existing implementation of this component.

4.2 PaaS: Abstract Layer

4.2.1 Registry

The Registry is a core component in N2Grid and N2Sky therefore there is a detailed interface specification. A neural network paradigm service is seen as a *Resource* (see *Domain model* in Subsection 3.3.3.2 and *Resource hierarchy* in Figure 3.5). A formal service description of each paradigm service is provided within a `NNServiceDescription` XML element listed in Appendix A (A.1) In case of unreachable services (i.e. the trigger method `checkRegisteredServices()` returns `false`) and after calling the method `removeFromRegistry()` the service will be removed from the *Registry*.

An overview of its Interface is shown in Table 4.4. The term *Neur.* within a column *mandatory* means that this method or parameter is only available for neural network paradigms and not for arbitrary services. Existing instances of the *Registry* component are:

- N2Sky Registry
- N2Grid Registry

4.2.2 Monitoring

An Interface definition of the still not integrated *Monitoring* component is shown in Table 4.5 according to the *Hyperic HQ Web Services API specification* [49]

4.2.3 SLA

The interface specification of the not yet implemented *SLA* component (described in Subsection 3.2.3) is shown in Table 4.6.

4.2.4 Controlling and Accounting

The interface specification of the not yet implemented *Controlling and Accounting* component (described in Subsection 3.2.4) is shown in Table 4.7.

Registry (mand.)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			*
	schemaLocation	URL, directory	*
	registryURL	URL	*
	searchableAttributes	String , sep.	*
	holdbackTime	Long (sec)	*
refreshTime < holdbackTime	refreshTime	Long (sec)	*
Trigger	currentTime = lastRefresh + refreshTime		
checkRegisteredServices()	removeTime = currentTime + holdbackTime		
Inbound interface		WS	*
getHoldbackTime()		synchr.	*
	holdbackTime	Long (sec)	*
getSchemaLocations()		syn.	*
OUT	schemaLocations	String[]	*
getAllServiceIDs()		synchr.	*
OUT	serviceIDs	Long[]	*
getAllNNServiceIDs()		synchr.	Neur.
OUT	NNserviceIDs	Long[]	Neur.
submitServiceDescription()		synchr.	*
IN	serviceID or name	Long/String	*
OUT	ServiceDescription	String	*
submitNNServiceDescription()		synchr.	Neur.
IN	serviceID or name	Long, String	Neur.
OUT	nnDescription	nnDescription	Neur.
getServiceIDsByParadigm()		synchr.	Neur.
IN	paradigm	String	Neur.
OUT	serviceIDs	Long[]	Neur.
getSearchCriteria()		synchr.	
OUT	searchCriteria	String[]	
searchServicesByCriteria()		synchr.	
IN	searchCriteria	String[]	
OUT	serviceIDs	Long[]	
getTrainingEndpoint()		synchr.	Neur.
IN	serviceID	Long	Neur.
OUT	serviceEndpoint	URL	Neur.
getEvaluationEndpoint()		synchr.	Neur.
IN	serviceID	Long	Neur.
OUT	serviceEndpoint	URL	Neur.
registerService()		synchr.	*
IN	acronym, name	String	*
IN	serviceDescription	String	*
IN	serviceEndpoint	URL	*
IN	searchCriteria	String[]	
IN	NNServiceDescript.	JSON/XML	*
removeFromRegistry()		synchr.	*
IN	serviceID	Long	*
Used components:	Component Archive		

Table 4.4: Interface specification of the *Registry component.*

Monitoring (P1)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			*
	registryURL	URL	*
Inbound interface		WS or API	*
alert			
application			
autodiscovery			
control			
dependendy			
event			
escalation			
resource			
role			
...			
Used components:	Registry		*
	Simulation Mgmt.		*
	Business Admin., ...		

Table 4.5: Interface specification of the *Monotoring component..*

SLA (P1)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			*
	registryURL	URL	*
Inbound interface		Web Service	*
createSLAContract()		synchr.	*
IN	userID	long	*
IN	packageID	long	*
IN	expiry	Date/Time	*
IN	SLALevel	String	*
OUT	SLAContract	SLAContract	*
getSLAContract()		synchr..	*
IN	contractID	long	*
OUT	SLAContract	SLAContract	*
searchSLAContracts()		synchr.	
IN	searchParams	String[]	
OUT	SLAContracts	SLAContract[]	
Used components:	Registry		*
	DBMS		

Table 4.6: Interface specification of the *SLA component.*

Controlling and Accounting (P1)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			*
	registryURL	URL	*
Inbound interface		WS or API	*
buyPackage()		synchr.	*
...			
bookSingleUseCosts()		synchr.	*
...			
calculateInvoice()		asynchr.	*
...			
bookPayment()		synchr.	*
...			
checkPayments()		asynchr.	*
...			
Used components:	Registry		*
	Access Control		*
	DBMS		

Table 4.7: Interface specification of the *Controlling and Accounting component.*

4.2.5 User and Role Management

An interface specification of the *Usermanagement* component described in Subsection 3.2.5 is shown in Table 4.8). The *N2Sky Usermanagement* is an existing instance of it.

Usermanagement (P1)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			*
	registryURL	URL	*
Inbound interface		WS or API	*
CRUD User()		synchr.	*
...			
CRUD Role()		asynchr.	*
...			
Used components:	Registry		*
	Access Control		*
	DBMS		

Table 4.8: Interface specification of the *Usermanagement component.*

4.2.6 Access Control

An Interface specification of the *Access Control* component described in Subsection 3.2.6 is shown in Table 4.9. Access control has not been integrated into N2Sky so far.

Access Control (P1)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			*
	registryURL	URL	*
Inbound interface		WS or API	*
CRUD Resource()		synchr.	*
...			
CRUD Privilege()		synchr..	*
IN	resourceID	long	*
IN	userID	long	*
IN	expirationDate	DateTime	
OUT	privilegeID	long	*
...			
Used components:	Registry DBMS		*

Table 4.9: Interface specification of the *Access Control component.*

4.2.7 Workflow System

Scientific workflow systems described in Subsection 3.2.7 themselves are often large systems using heterogenous interfaces so it has not been implemented in N2Sky until now. A basic interface specification of the *Workflow system* component is shown in Table 4.10.

Workflow System (P2)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			*
	registryURL	URL	*
Inbound interface		Web Service	*
Process Mgmt.		synchr.	*
...			
Instance Mgmt.		synchr./asynchr..	*
Controls the individual process instances to manage concurrency.			
Used components:	Registry		*
	Simulation Services		*
	...		

Table 4.10: Interface specification of the *Workflow system component.*

4.2.8 Knowledge Management

A basic interface specification of the *Knowledge Management* component is shown in Table 4.11.

Knowledge Management (P2)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			
Inbound interface		WS, or API	*
CRUD operations over SPARQL queries		synchr.	*
Used components:	Data Archive		

Table 4.11: Interface specification of the *Knowledge Management component.*

4.2.9 Component Hosting Platform

A basic interface specification of the *Component Hosting Platform* described in Subsection 3.2.9 is shown in Table 4.12. A N2Sky integration doesn't exist so far.

Component Hosting Platform (P2)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			*
	registryURL	URL	*
Inbound interface		Web Service	*
getAvailableServer()		synchr.	*
startServer()		asynchr.	*
stopServer()		asynchr.	*
deploy()		asynchr.	*
undeploy()		asynchr.	*
getDeploymentURL()		synchr.	*
getAvailableDBMS()		synchr.	*
...			
Used components:	Registry		*
	Server		*
	DBMS		*
	Knowledge Base		

Table 4.12: Interface specification of the *Component Hosting Platform.*

4.2.10 Annotation Service

An Interface specification of the *Annotation Service* described in Subsection 3.2.10 is shown in Table 4.13. So far, there is no existing implementation.

Annotation Service (P2)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			
Inbound interface		WS or API	*
createAnnotation()		synchr.	*
IN	forObjectID	long	*
IN	text	String	*
IN	annotationType	String	*
IN	attachment	File	
OUT	annotationID as objectID		*
getAnnotations()		synchr.	*
IN	forObjectID	long	*
OUT	annotations	Annotation[]	*
editAnnotation()		synchr.	*
IN	annotationID	long	*
IN	changedText	String	*
OUT	changedText	String	*
deleteAnnotation()		synchr.	*
IN	annotationID	long	*
OUT	annotationID	long	*
Used components:	Knowledge Base		

Table 4.13: Interface specification of the *Annotation Service component.*

4.3 PaaS: Neural Network Layer

4.3.1 Simulation Services

Paradigm Simulation Services consist of methods described in Subsection 4.3.1.2 and provide training and evaluation of neural objects which in turn are instances of neural network paradigms. In N2Sky each paradigm service has to provide these methods in a strict manner except `createNeuralObject()`. Since object creation is similar for each paradigm we added this functionality to the *Simulation Management* service. Table 4.15 shows the interface specification of *Simulation Services.*

Existing instances (to name only a few):

- Backpropagation 01
- Kohonen 02
- Perceptron 01

4.3.1.1 Types

Table 4.14 shows the list of neural network data types and their corresponding schema files used by operations of the *Simulation Service*.

Type	Schema file
nnDescription	n2sky-description.xsd (A.1)
	The design parameters of the neural net.
nnDefinition	n2sky-definition.xsd (A.2)
	Parameters and input values of the neural net.
nnData	n2sky-data.xsd (A.3)
	The input data for training or evaluation.
simulationResult	n2sky-result.xsd (A.4)
	The results of training or evaluation operations.

Table 4.14: N2Sky Schema types.

4.3.1.2 Operations

- **getServiceDescription**(). The operation `getServiceDescription` returns a neural network description (`nnDescription`) of the provided neural network service and has no input parameter.
- **createNeuralObject**(). The operation `createNeuralObject` is used to create a new instance of a neural network paradigm or to duplicate an existing neural object.
- **train**(). The operation `train` is used to train a new instantiation or to further adjust an already trained neural object. The operation returns a `submission-ID` as `String` for later retrieving of a trained neural network. Parameters are the neural network definition (`nnDefinition`) and neural network data (`nnData`: text-based or binary). The unique callback URL is composed of the URL to the task management service or a workflow system followed by a `submission-ID`.
  - **evaluate**(). By the operation `evaluate` an already trained neural network (i.e. a trained neural object compliant to `nnDefinition`) can be sent to this service method, accompanied by evaluation data (`nnData`). This operation returns a `submission-ID` as `String` for later retrieving result data. The unique callback URL is composed of an URL to the task management service or a workflow system followed by a `submission-ID`.
- **checkStatus**(). The operation `checkStatus` provides information about a submitted training or evaluation. The only one input parameter is a `submission-ID` as `String`. It returns an `integer` number representing the progress with the following semantic:
  - `-1`: error, `0` to `99`: percentage of progress, `100`: finished.

- **getResult**(). The operation `getResult` retrieve result data. The result schema contains a trained network and other information about the training or only output data and other information in the case of an evaluation. The only one input parameter of this operation is a `submission-ID` as `String`. A client is able to save the network for later evaluation runs.

- **getNNDefinition**(). The operation `getNNDefinition` returns a neural network definition (`nnDefinition`) of the neural object. The only one input parameter is a `submission-ID` returned by those operations: `createNeuralObject, train` and `evaluate`.

Simulation Service (mand.)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			*
	schemaLocation	URL, directory	*
	registryURL	URL	*
Inbound interface		Web Service	*
getServiceDescription()		synchr.	*
OUT	nnDescription	nnDescription	*
train()		asynchr.	*
IN	nnDefinition	nnDefinition	*
IN	nnData	nnData	*
IN	SLA	String {A|B|C}	
IN	callbackURL	URL, unique	*
instant output	submissionID	String	*
evaluate()		asynchr.	*
IN	nnDefinition	nnDefinition	*
IN	nnData	nnData	*
IN	SLA	String {A|B|C}	
IN	callbackURL	URL, unique	*
instant output	submissionID	String	*
checkStatus()	for train(), evaluate()	synchr.	*
IN	submissionID	String	*
OUT	status	int	*
getResult()		synchr.	*
IN	submissionID	String	*
OUT	simulationResult	simulationResult	*
getNNDefinition()		synchr.	*
IN	submissionID	String	*
OUT	nnDefinition	nnDefinition	*
Callback to operations:	train(), evaluate()	Web Service	*
sendResult()		asynchr.	*
IN	simulationResult	simulationResult	*
Used components:	Registry: registerService(), removeFromREgistry()		

Table 4.15: Interface specification of a *Simulation Service*.

4.3.2 Simulation Management

We see *Simulation Management* and *Business Administration* componenta as central entry point for user interface components in *SaaS Service Layer* to underlying components. An overview of its Interface specification is shown in Table 4.16. The *N2Sky Simulation Management* is one existing instance of the *Simulation Management* component described in Subsection 3.3.2.

Simulation Management (mand.)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			*
	registryURL	URL	*
Inbound interface		Web Service	*
logIn(), logOut()		synchr.	*
...			
getSubscriptions()		synchr.	*
...			
getSubscribeableParadigms()		synchr.	*
...			
getCreations()		synchr..	*
...			
getTrainings/Evaluations()		synchr..	*
...			
createNeuralObject()		synchr.	*
...			
train/retrain/evaluate()		asynchr.	*
...			
checkStatus()		synchr.	*
...			
getResult()		asynchr..	*
...			
Used components:	Simulation Services		*
	Registry		*
	Usermanagement		*
	...		

Table 4.16: Interface specification of the *Simulation Management component.*

4.3.3 Business Administration

An overview of the Interface specification od the *Business Administration* component described in Subsection 3.3.3 is shown in Table 4.17. So far, there is no instance implemented.

Business Administration (P1)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			*
	registryURL	URL	*
Inbound interface		Web Service	*
registerUser()		synchr.	
...			
integrateParadigm()		asynchr.	*
...			
integrateHardware()		synchr.	
...			
buyPackage()		synchr..	*
...			
debitAccount()		synchr.	*
...			
creditAccount()		synchr.	*
...			
createEndUserBill()		asynchr.	*
...			
checkAccounts()		asynchr..	*
...			
Used components:	Registry		*
	Controlling and Accounting		*
	Usermanagement		
	...		

Table 4.17: Interface specification of the *Business Administration component.*

4.3.4 Hosted Components

A rudimentary Interface specification is shown in Table 4.18. Since the *Component Hosting Platform* is not yet implemented, there are also no *Hosted Components.*

Hosted Components (P2)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			*
	registryURL	URL	*
Inbound interface		WS, API	*
Hosted Component functionality			
...			
Used components:	Component Hosting Platform		*
	DBMS		

Table 4.18: Interface specification of *Hosted Components.*

4.4 SaaS: Service Layer

4.4.1 Query Interface

A basic Interface specification of the *Query Interface* described in Subsection 3.4.1 is shown in Table 4.19. So far, there is no existing implementation for it.

Query Interface (P1)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			*
	knowledgeBaseURL	URL,	*
	registryURL	URL	*
Inbound interface		Web	*
searchForProblem()		synchr.	*
IN	problemName	String	*
OUT	proposedParadigms	nnDescription[]	*
searchForSolution(()		synchr.	*
IN	paradigmName	String	*
OUT	paradigm	nnDescription[]	*
Used components:	Knowledge Base		

Table 4.19: Interface specification of the *Query Interface component.*

4.4.2 Web Portal

A basis Interface specification of the *Web Portal* is shown in Table 4.20. It includes both functionality of the *Smartphone App* and additional *Business Use Cases* depicted in Subsection 3.3.3.

Web Portal (P1)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			*
	schemaLocation	URL, directory	*
	registryURL	URL	*
Inbound interface		Web	*
createNeuralObject()		synchr.	*
train()		asynchr.	*
retrain()		asynchr.	*
evaluate()		asynchr.	*
showStatus()		synchr.	*
...			
manageUserAccount()		synchr..	*
createEndUserBill(()		asynchr.	
checkAccounts()		asynchr.	
...			
Used components:	Registry		*
	Simulation Management		*
	Business Administration		*
	Query interface		

Table 4.20: Interface specification of the *Web Portal.*

4.4.3 Smartphone App

A basic Interface specification of the *Smartphone app* is shown in Table 4.21. The existing *N2Sky Smartphone app* as an iOS build of the *Mobile Web Portal* primarily acts as a graphical user interface of the *Simulation management* component.

Smartphone App (mand.)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			*
	schemaLocation	URL, directory	*
	registryURL	URL	*
Inbound interface		Web	*
createNeuralObject()		synchr.	
...			
train()		asynchr.	*
...			
duplicateObject()		synchr.	
...			
evaluate()		asynchr..	*
...			
showStatus()		synchr.	*
...			
Used components:	Registry		*
	Simulation Management		*
	Business Administration		
	Query interface		

Table 4.21: Interface specification of the *Smartphone App.*

4.4.4 Hosted UIs

A basic interface specification of a *Hosted UI* is shown in Table 4.22. A *Hosted UI* can be seen as a graphical user interface tor a *Hosted component* running on the *Component hosting Platform.*

Hosted UI (P2)			
Interface, operation, I/O	parameter name	param. type	mand.
Configuration			*
	registryURL	URL	*
Inbound interface		Web	*
Hosted UI functionality			
...			
Used components:	Hosted Component		*

Table 4.22: Interface specification of a *Hosted UI component.*

Chapter 5

Tutorial

This tutorial describes how to create a simple Backpropagation neural network to solve the well-known XOR-problem.

5.1 Paradigm Selection

Problem: The logical operation *exclusive disjunction*, is a type of logical disjunction on two operands that results in a value of "true" if and only if exactly one of the operands has a value of "true".

Solution: Using a 2-5-1 *Backpropagation* neural network. To solve the XOR-problem we use a fully connected Backpropagation net with 3 layers. The input layer contains 2 input neurons, the hidden layer holds 5 neurons, and finally the output layer encloses only a single neuron.

The user selects the *BPROTUTO Tutorial Backpropagation* paradigm and instantiates a new neural net object based on this paradigm. Subsection 6.1.2 describes how to subscribe for a paradigm.

5.2 Neural Object Creation

Neural object creation is divided into two steps: *Layer* and *parameter* definition. Created objects are stored in SQL table *Creation*.

5.2.1 Layer Definition

Figure 5.1 depicts *input* and *output layer definition* by specifying the number of neurons within a paradigm-specific range (a), hidden layer definition by specifying layer name and number of hidden neurons (b) and connecting layers with *Full Connections* by tapping the particular layer (c). Each form contains practicable default values.

Figure 5.1: Structure definition for a neural object with 2 input, 3 hidden and 1 output neuron whereas hidden layer is full connected with output layer.

Figure 5.2: Parameter definition and resulting parameter and structure descriptions.

5.2.2 Parameter Definition

There are three types of parameters:

1. value,
2. boolean
3. and combo parameters

Specify a desired numerical value for value parameters, tap the boolean parameter to turn it on or off and select a predefined entry for a combo parameter. Figure 5.2 depicts value parameter (P), boolean parameter and combo parameter (CP) definition (screenshot a), a finished parameter definition (b) and a finished structure definition including layers and connections (c). These infos can be retrieved by tapping the *Creation info* button.

5.3 Training

After layer and parameter definition the resulting neural object (Creation) can be trained and - if supported by the paradigm - retrained (i.e. to optimize a previous training with additional samples). However, for every creation several separated trainings can be appended. To input training patterns (and in the next step evaluation data sets) there are three options:

1. Type in or copy/paste integer or floating point values
2. Get inputs from the data archive (tap the *Get samples* button)
3. Query data using SQL or NoSQL (JSON-style) query statement for document-based databases.

CSV is the most basic style for input samples because it includes minimal overhead. Numerical values separated by semicolons are entered line per line. Every line represents exactly one input sample. In case of training patterns, CSV inputs are not separated explicitely from its corresponding outputs as they are in JSON and XML format. For this reason, metadata is displayed on top of it. In this case in1; in2; out1; indicates 2 input and 1 output value, a semicolon at the end of each line may be omitted. Figure 5.3 depicts CSV training data separated by semicolons (a), a list of current trainings - the latest with a progress status of 25% (b) and training metadata showing an error rate diagram (c). Training (and evaluation) samples can also be expressed in JSON or XML in a matrix syntax (rows and columns).

By tapping the *Train* button a new training task is started in the Cloud and training metadata is shown. For refreshing the progress indicator tap the *Refresh* button. You don't have to wait until training is finished. If you want you can start another task or log out. The started training task will continue to run unaffected thereof. This training object will appear on top of the *Current trainings* list showing a chronological overview of the latest trainings you have initiated. After training completed successfully, training

results (calculated weights and result values, etc.) and an error graph are shown and stored in a database for further usage (e.g. evaluation).

Figure 5.3: CSV sample data input, a running training and error rate diagram.

5.4 Evaluation

The last step for problem solution is to evaluate the previously trained neural object . By tapping the trained neural network an empty list of evaluations appears. After tapping *New evaluation...* you also have the possibility to define the evaluation data by an explicit list or a query statement. The difference between training and evaluation is that evaluation data contains only inputs (INPUTMATRIX) as shown in Figure 5.4 (a), the resulting samples are in the same style as training results (b). These screenshots depict a session where a desktop browser (Safari) is employed instead of an iPhone browser (Mobile Safari).

After a particular evaluation task is performed in the Cloud the finalized evaluation object is persisted in the data store and appears both in your subscription tree and in the *Current evaluations* list. This solution to the given problem then is accessible from everywhere and can be further processed by other systems.

Numerical result samples as shown in Figure 5.4 (b) always have the following structure:

- **SAMPLE**. A single training pattern or evaluation data set containing input and output matrices/vectors. In JSON format these samples are combined into an array which is a child element of `SAMPLE`.
- **INPUT**. The sample's input matrix/vector.
- **OUTPUT**. The sample's output matrix/vector.
- **ROW**. A row of a matrix containing column (C) elements.
- **C**. A single numerical value.

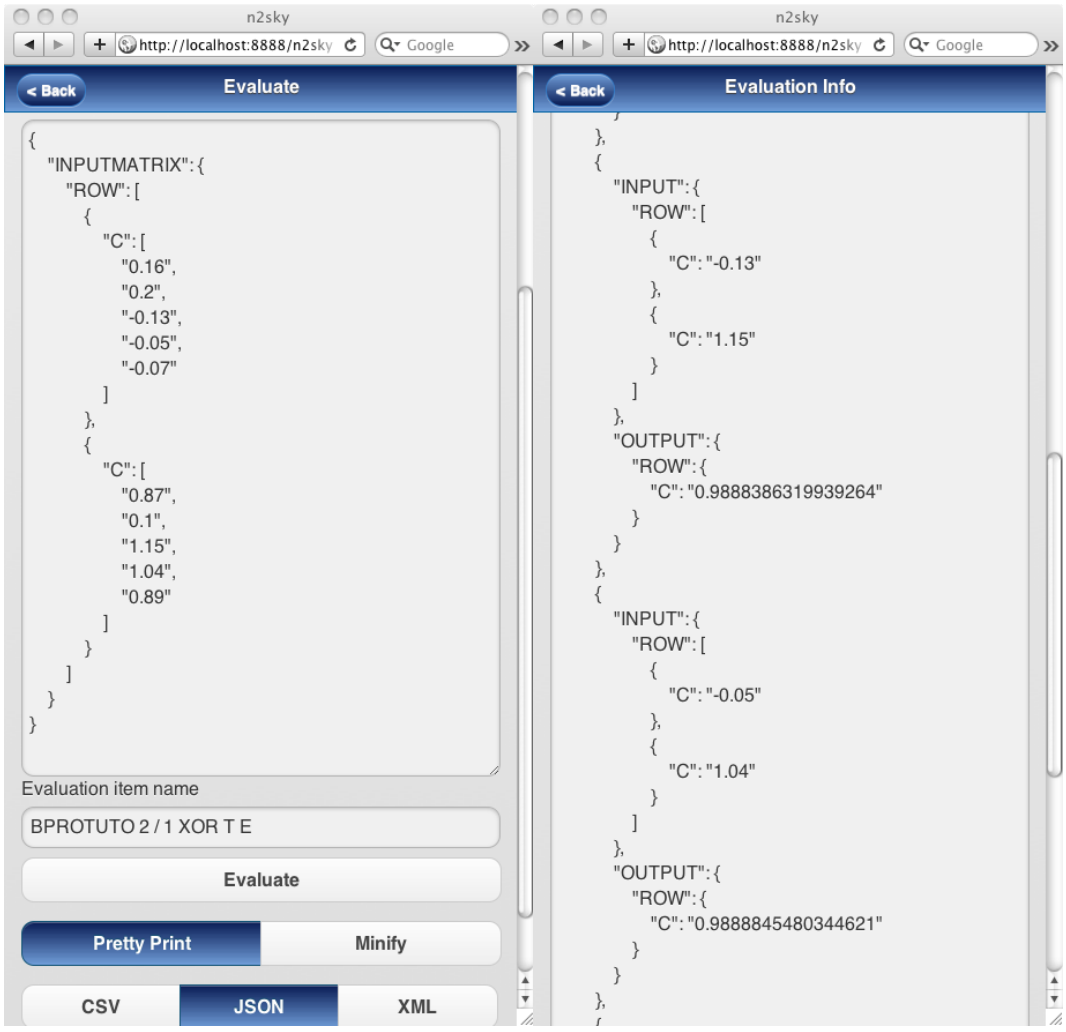

Figure 5.4: XOR evaluation input and result JSON data.

Chapter 6

User Guide

This chapter is divided into two sections according to N2Sky's major stakeholders: *End User* and *Administrators* whereas the next chapter is all about component developers.

6.1 End User Guide

6.1.1 Login

Authentication via username and password is mandatory to identify the stakeholder and to manage access control. If login proceeded successfully all system-wide attached database connections are available per default for any user. Figure 6.1 depicts a login screen (a), the registration page for new users (b) and the subscription list as main page after a successful login (c).

6.1.2 Paradigms and Subsscriptions

Paradigms are offered as subscribeable services and can be queried using SQL with following criteria:

- id
- acronym
- name
- drscription
- type
- service_path

Figure 6.2 shows such searchable metadata (a), a subscription form (b) and a formal paradigm description according to the Vienna Neural Network specification language

Figure 6.1: Login, registration form and list of subscriptions as simulation tree root.

Figure 6.2: Paradigm metadata, subscription form and paradigm service description.

(VINNSL) [8] stored as XML in the field description_file. Paradigms are listed in SQL table *Resource* with type *PA* because there are also paradigm groups with type *PG*. Some paradigms can be used for free without subscription but neural objects created without subscription are not shown in yout personal simulation tree, they can only be queried using SQL.

6.1.3 Datastream Queries

6.1.3.1 Concept

A highlight of the N2Sky system is the use of standardized and user-friendly database query languages both for administration tasks and neural network simulation tasks whereas the user applies the same tool based on a homogenous interface. Administrative tasks like searching for neural network paradigms and objects retrieve metadata from our domain model mapped to relational database systems. During neural network simulation tasks the functional data stream definition described by SQL and NoSQL allows to specify real world training and evaluation data sets in a comfortable and natural way on a global scale. Training and evaluation data sets can be huge data volumes, therefore this functional specification by a query language statement is extremely comfortable. This unique feature of N2Sky allows for combining globally stored, distributed data easily.

6.1.3.2 Syntax

To implement the datastream concept we specified a query syntax listed below (Listing 6.1). This interface is intended currently only to read data, the http method GET corresponds to the SELECT expression in SQL. Such queries can be fired during training, retraining or evaluation preparation (as depicted in Figure 6.3) by either tapping "Query data" or "Train/Retrain/Evaluate" to send your query directly to the chosen paradigm service.

Listing 6.1: Datastream query syntax in EBNF notation

```
1 method      = "GET";
2 path        = "http", { char };
3 selector    = "_find?criteria={", { char }, "}" |
4               "selector={", { char }, "}" |
5               an arbitrary selector;
6 url         = path, [ "?", selector }, { "&", selector };
7 exclusion   = "--exclude '", regex, "'";
8 query       = method, url, { exclusion };
```

6.1.3.3 MySQL Simulation Management DB

The *Simulation management DB* contains all N2Sky monitoring and business data described in Subsection 3.3.3.2 (domain model). Furthermore some simulation input and result data can be queried from the following tables:

- **nndata.** Training or evaluation input samples
- **nnresult.** Training or evaluation result samples generated by paradigms
- **data_resource.** paradigm-specific input test data specified by the paradigm developer
- **data_creation.** creation-specific input data stored using the *Store samples* button.

Figure 6.3 depicts a SQL query (*file* contains a XML string) (a) and a MongoDB query using the syntax described above (b). Database credentials are managed automatically by the N2Sky system (table *nndb*). These two screenshots demonstrate how to specity training data sets via SQL and NoSQL queries. For accessing large training or evaluation data sets at big data stores the NoSQL approach performs better due to improved scalability of NoSQL key/value stores. For administrative tasks (choosing neural network paradigms, trained neural network objects or any business metadata) purely SQL queries are used because of our domain model is mapped onto relational database systems.

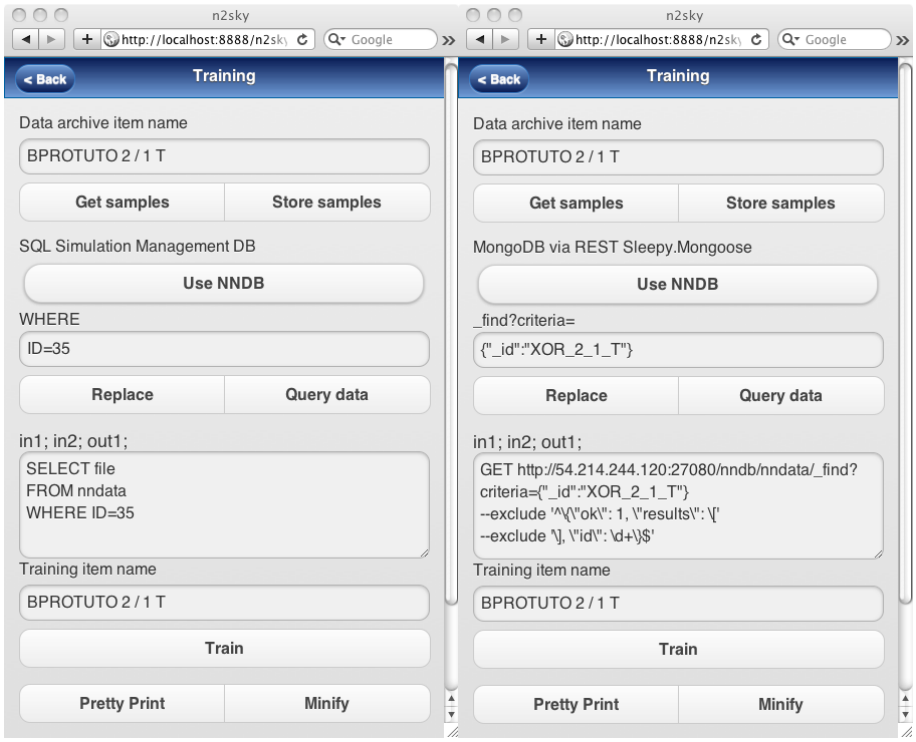

Figure 6.3: SQL and MongoDB data query.

Listing 6.2: Inserting JSON data using user-defined ID via *mongo* shell.

```
1  >>$ mongo
2  > use nndb
3  > db.auth( <user>, <password> )
4  > t = { "_id" : "XOR_2_1_T", "INPUTMATRIX" : { "ROW" : [ { "C" : [ "0.0", ...
5  {       "_id" : "XOR_2_1_T",
6          "INPUTMATRIX" : {
7                  "ROW" : [
8                          {
9                                  "C" : [
10                                         "0.0",
11                                         "0.0",
12                                         "1.0",
13                                         "1.0"
14                                 ]
15                         },
16                         {
17                                 "C" : [
18                                         "0.0",
19                                         "1.0",
20                                         "0.0",
21                                         "1.0"
22                                 ]
23                         }               ]
24                 },
25         "OUTPUTMATRIX" : {
26                 "ROW" : {
27                         "C" : [
28                                 "0.0",
29                                 "1.0",
30                                 "1.0",
31                                 "0.0"
32                         ]
33                 }
34         }
35 }
36 > db.nndata.insert( t )
```

6.1.3.4 MongoDB

MongoDB [50] is a scalable open source (AGPL license) NoSQL document-based store written in C++ with some friendly features of SQL like indexes and queries. It is designed for storing big data on 64 bit systems, an empty database takes up 192 MB and on 32 bit systems address space is limited to 2.5 GB.

The code snippet 6.2 shows how to connect to our NN database, how to authenticate (line 4) and how to insert valid simulation input data by using an user-defined ID (line 6) instead of an *ObjectID* automatically generated by MongoDB. This enables simple human-readable queries - this user-defined ID also has to be unique. To be more flexible we use provided web service interfaces instead of integrating in Java directly. So arbitrary databases can be used unless REST query interfaces are provided.

6.1.3.4.1 Sleepy.Mongoose for MongoDB Sleepy.Mongoose [51] is a MongoDB http interface written in Python. Sleepy.Mongoose results are always enveloped within status information so we have to exclude this if we want to use results directly as input data for paradigm services. Such a sample query is listed below (6.3).

Listing 6.3: MongoDB query via Sleepy.Mongoose with regex for excluding metadata.

```
1 GET
2 http://192.168.56.101:27080/nndb/nndata/_find?criteria={"_id":"XOR-2-1-T"}
3 --exclude '^\{\"ok\": 1, \"results\": \['
4 --exclude '\], \"id\": \d+\}$'
```

Listing 6.4: MongoDB query via Sleepy.Mongoose with curl.

```
1 >>$ host=http://A.B.C.D
2 >>$ mongoose=$host:27080
3 >>$ db=nndb
4 >>$ connect=$mongoose'/$db/_connect'
5 >>$ auth=username=<username>&password=<password>
6 >>$ curl --data $auth $connect
7 >>$ collection=nndata
8 >>$ insert=$mongoose'/'$db'/'$collection'/_insert'
9 >>$ json='[{"x":1}]'
10 >>$ curl --data 'docs='$json $insert
11 >>$ escape('{"x":1}')
12 %7B%22x%22%3A1%7D
13 >>$ criteria='_find?criteria=%7B%22x%22%3A2%7D'
14 >>$ curl -X GET $mongoose'/'$db'/'$collection'/'$criteria
15 # equivalent to GET query in browser:
16 http://A.B.C.D:27080/nndb/nndata/_findCriteria=%7B%22x%22%3A2%7D
```

Listing 6.4 describes a sample session including connecting to a database, authentication (lines 1 to 6), inserting JSOn data into a collection and finally constructing a query to fetch this data.

6.1.3.4.2 DrowsyDromedary for MongoDB DrowsyDromedary [52] is an open source REST interface for MongoDB written in Ruby. A sample query during training / retraining / evaluation is listed below (6.5) whereas sample input data with _id "XOR-2-1-T" will be retrieved. Braces and quotes in the URL will be encoded from N2Sky automatically.

A *DrowsyDromedary* query string is composed as follows:

1. **GET.** The http method `get`.
2. **host/nndb.** Our MongoDB neural network databas `nndb`.
3. **nndata.** The **name** of the input sample data collection.
4. **selector={key: value}.** *DrowsyDromedary's* syntax for querying JSON data.

Listing 6.5: MongoDB query via DrowsyDromedary.

```
1 GET http://192.168.56.101:9292/nndb/nndata?selector={"_id":"XOR-2-1-T"}
```

6.2 Administration Guide

6.2.1 Virtualization

If you want to create and test your own image on QEMU/KVM or VirtualBox locally instead of customizing an Eucalyptus sample image you have to do the following steps:

1. If your Eucalyptus infrastructure contains nodes supporting only images that are smaller or equal 2 gigabytes (<= 2G) and you want to run your image on it, you have to create such `.raw` image with QEMU.
2. If you want to run this image in VirtualBox, you have to convert the `.raw` image to a VMWare VMDK format by using the `VBoxManage` command line tool which is part of VirtualBox as performed in Listing 6.6.
3. Use this VMDK image in VirtualBox and install Debian. The size of a swap partition does not matter because it will be extracted during cloudification, only your root partition (put system, applications and data on exactly one partition) has to be <= 2 GB.
4. Install Debian including OpenSSH and Apache but without any graphical user interface to save disk space and RAM.

We have two Debian systems, one physical system on which QEMU, VirtualBox, euca2ools and other tools are installed. The "virtual" system is our Debian image running on VirtualBox and will be deployed on Eucalyptus. Listing 6.6 shows how to create a *raw* image with KVM and how to convert it to a VMDK image to use it in VirtualBox.

Listing 6.6: Image creation and conversion.

```
1 # physical
2 >>$ kvm-img create -f raw debi.img 2G
3 >>$ VBoxManage convertfromraw --format VMDK debi.img debi.vmdk
```

We use Debian 6 in a 64 bit version because it is stable and doesn't need much disk space. It also runs within an Eucalyptus infrastructure using a Xen kernel for performance optimization. Listing 6.7 shows Debian package management commands installing additional packages needed to run N2Sky:

1. A Java Runtime Environment (>= Java SE 6)
2. Tomcat 6 (or an alternative Java web application container)
3. Tomcat's administration application for deploying and removing webapps remotely over a web interface.
4. MySQL server and client or another RDBMS (you only have to change database.properties within your JPA configuration).
5. PHP 5 and phpMyAdmin (only for MySQL administration).

Listing 6.7: Post-Installs.

```
1 # virtual
2 >># apt-get install default-jre
3 >># apt-get install tomcat6
4 >># apt-get install tomcat6-admin
5 >># apt-get install mysql-server mysql-client
6 >># apt-get install php5 libapache2-mod-php5 phpmyadmin
7 >># /etc/init.d/apache2 restart
```

6.2.2 Cloudification

6.2.2.1 Image Preparation

The first and essential cloudification step is to delete Debian's network rules file on the virtual image as shown in Listing 6.8.

Listing 6.8: Deleting network rules.

```
1 # virtual
2 # Trap: Delete network rules and shut down
3 >>$ rm -rf /etc/udev/rules.d/70-persistent-net.rules
```

Convert VMDK image to *raw* format (`.img`). Hint: Use the original VMDK image, otherwise you will get an error message (Cannot register hard disk) if you use a copy of it. Listing 6.9 shows an in-depth example.

Listing 6.9: Converting a VirtualBox image to raw (i.e. .img) format..

```
1 # physical
2 >>$ VBoxManage clonehd --format RAW debi-copy.vmdk debi.img
3 VBoxManage: error: Cannot register the hard disk
4  '.../debi-copy.vmdk'.
5  '.../debi.vmdk' with UUID {31b...} already exists.
6 >>$ VBoxManage clonehd --format RAW debi.vmdk debi.img
7 0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
8 # As an alternative converting from dynamically growing VDI
9 >>$ VBoxManage clonehd --format RAW debi.vdi debi.img
```

A very important step in cloudification of a QEMU or VirtualBox image is to extract data partition as described in [53]. The resulting image then only contains that partition without master boot record and swap partitions, otherwise Eucalyptus is not able to start those images. Listing 6.10 shows a session preparing our N2Sky image.

1. Start `parted` (a Linux partitioning tool, it doesn't run on Mac OS X), set partition table unit to byte., print partition table and quit *parted*
2. Use `dd` to extract data partition and take all byte values from this partition table.

Listing 6.10: Extracting data partition using *parted* and *dd*.

```
>>$ parted debi.img
> unit b
> print
# partition table is shown (unit byte)
> quit
>>$ dd if=debi.img of=n2sky.img bs=512 skip=2048 count=3880960
# End of master boot record / block size
# Number of blocks to skip: 1048576 / 512 = 2048
# (End - start of data partition) / block size
# Number of blocks to read: (1988100095 - 1048576) / 512 = 3880960
```

6.2.2.2 Eucalyptus Deployment

Listing 6.11 shows a sample session describing an Eucalyptus image/instance lifecycle. A detailed description is shown in the Google Summer of Code Tutorial [53]. We tested euca2ools both on Debian 6 and Mac OS X 10.6. By running the `eucarc` file all needed environment variables are set needed to connect to your Eucalyptus installation. You have the choice to bundle kernel and ramdisk files or set references to even uploaded kernel and ramdisk images. Check if your kernel is compatible to your Eucalyptus infrastructure and use a Xen kernel to get optimal system performance. If you want to choose another kernel to run an already uploaded image, you have the possibility to define them as arguments of the *euca-run-instances* command.

Listing 6.11: A sample session describing an Eucalyptus image/instance lifecycle.

```
# physical
# Credentials: Set environment variables
>>$ source ~/.euca/eucarc
>>$ euca-describe-availability-zones
# $kernel = kernel in /boot of virtual image
# $ramdisk = ramdisk in /boot of virtual image
# $arch = "i386"|"x86_64" # 32 or 64 bit
>>$ euca-bundle-image -i n2sky.img --kernel $kernel --ramdisk $ramdisk
    --arch §arch
# Generating manifest /tmp/n2sky.img.manifest.xml
>>$ euca-upload-bundle -m /tmp/n2sky.img.manifest.xml -b n2sky-bucket
>>$ euca-register n2sky-bucket/n2sky.img.manifest.xml
# Registration successful if image ID was created
>>$ euca-describe-images | grep n2sky
>>$ euca-run-instances <emi-xxx>
# Successfully booted if instance ID created and IP address set
>>$ euca-describe-instances | grep n2sky
# Shutdown instance
>>$ euca-stop-instances <i-xxx>
# Delete bundle
>>$ euca-deregister <emi-xxx>
>>$ euca-delete-bundle -b n2sky --clear
```

6.2.3 Object-Relational Mapping (ORM)

N2Sky's domain model (described in Subsection 3.3.3.2) is implemented using Spring Data JPA as a lightweight implementation of the Java Persistence API. ORM is managed by two configuration files in `/src/main/resources/META-INF`: `persistence.xml` (see Listing 6.12 and `database.properties` for holding individual credentials. The transaction type "`RESOURCE_LOCAL`" is always the right choice except if you are using JTA both for connections to databases on localhost as on remote hosts. Spring uses *Hibernate* as underlying persistence technology so various *Hibernate* properties are set. For a production system please ensure that the value of `"hibernate.hbm2ddl.auto"` (line 6) is set to "`validate`", this means that no database schema changes can be made automatically by the Spring Framework.

Listing 6.12: The MySQL persistence unit in persistence.xml.

```
1  <persistence-unit name="persistenceUnit" transaction-type="RESOURCE_LOCAL">
2    <provider>org.hibernate.ejb.HibernatePersistence</provider>
3    <properties>
4      <property name="hibernate.dialect"
5                value="org.hibernate.dialect.MySQL5InnoDBDialect"/>
6      <property name="hibernate.hbm2ddl.auto" value="validate"/>
7      <property name="hibernate.ejb.naming_strategy"
8                value="org.hibernate.cfg.ImprovedNamingStrategy"/>
9      <property name="hibernate.connection.charSet" value="UTF-8"/>
10   </properties>
11 </persistence-unit>
```

6.2.4 DBMS

6.2.4.1 MySQL

We decided to use MySQL as target relational database management system (RDBMS) for our ORM because MySQL is the most popular RDBMS in the Cloud and version 5.6 includes special Cloud replication features described in chapter 14 of O'Reilly's book *MySQL High Availability* [54]. For N2Sky several architectures are possible:

1. One MySQL Master DB on each Cloud instance replicated to one single MySQL slave DB outside the Cloud on real hardware (works well since MySQL 5.6).
2. Each Cloud image uses one single MySQL DB outside the Cloud (Disadvantage: Concurrent requests).
3. Using MySQL Cluster (Disadvantage: high configuration effort).

6.2.4.2 MongoDB

MongoDB is a NoSQL document-based store using JSON for semi-structured data storage. It is recommended to install the 64bit version because the 32bit one is limited to

2 GB of data. A sample installation on Debian 6 looks like Listing 6.13. The `10gen` package contains the latest stable release. As installation preparation import the `10gen` public GPG key by typing the command in line two. During installation the `mongodb` daemon will be configured so that MongoDB server starts automatically after system boot. `Mongo` is the interactive command-line client for MongoDB. and by tping `use` you can switch between databases, if it doesn't exist, it will be created automatically. After an unexpected shutdown you have to remove the `mongod.lock` file (see last line) after that MongoDB starts without any exceptions.

Listing 6.13: Installing MongoDB on Debian.

```
1  # Configure Package Management System (APT)
2  >># apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10
3  # Create a /etc/apt/sources.list.d/10gen.list
4  # file and include the following line for the 10gen repository:
5  # /etc/apt/sources.list.d/10gen.list
6  deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen
7  # Installing
8  >># apt-get install mongodb-10gen
9
10 # Collectionspace: /var/lib/mongodb
11 # Logs: /var/log/mongodb
12 # Starting: /etc/init.d/mongodb start
13 # Terminating: /etc/init.d/mongodb stop
14
15 # Interactive command line client
16 >>$ mongo
17 > use nndb
18
19 # Troubleshooting: Exception after unexpected shutdown:
20 >># cd /var/lib/mongodb/
21 >># rm mongod.lock
```

6.2.4.2.1 DrowsyDromedary. DrowsyDromedary [52], a REST interface for MongoDB is written in Ruby and therefore *rubygems* and *bundler* have to be installed before running DrowsyDromedary as shown in Listing 6.14. The start script in `/etc/init.d/` executes Bundler's `rackup` command in the directory `DrowsyDromedary-master`. In a shell script you have to prefix it with `bundle exec` otherwise you will get a runtime exception. For a production deployment you should deploy DrowsyDromedary with *Passenger* on Apache as described in [52],

6.2.4.2.2 Sleepy.Mongoose. Sleepy.Mongoose [51], a further REST interface for MongoDB is written in Python and therefore needs a Python environment as shown in Listing 6.15. After installing Python you have to install `mongo-python-driver` and then download source files from github. To start Sleepy.Mongoose, you change to this source directory (in our case `/opt/sleepymongoose/sleepy.mongoose-master/`) and run `python httpd.py` (`python` calls Python's interpreter).

Listing 6.14: Installing and starting DrowsyDromedary on Debian.

```
# Installing git, rubygems and bundler
>>#
        apt-get install git
        apt-get install rubygems
        apt-get install ruby-rack # to run 'rackup'
        gem install rubygems-update
        gem install bundler
#
# Download DrowsyDromedary source files
# Startup
# /etc/init.d/drowsydromedary start
...
NAME=DrowsyDromedary
DESC="MongoDB Ruby REST server"
# export PATH=/var/lib/gems/1.8/bin:${PATH}

case "$1" in
start)
  echo "Starting $DESC" "$NAME"
  cd /opt/drowsydromedary/DrowsyDromedary-master
  bundle exec rackup
  ;;
...

# executable script
>># chmod 755 /etc/init.d/drowsydromedary
```

Listing 6.15: Installing and starting Sleepy.Mongoose on Debian.

```
# Installing python-pkg-resources and python-setuptools
>># apt-get install python-setuptools
>># apt-get install build-essential python-dev
# Installing mongo-python-driver
>># easy_install pymongo
#
# Download Sleepy.Mongoose source files from github
# Startup
# /etc/init.d/sleepymongoose start
...
NAME=Sleepy.Mongoose
DESC="MongoDB http REST interface"
case "$1" in
start)
  echo "Starting $DESC" "$NAME"
  cd /opt/sleepymongoose/sleepy.mongoose-master/
  python httpd.py
  ;;
...

# executable script
>># chmod 755 /etc/init.d/sleepymongoose
```

6.2.5 Cronjobs

An easy way of running scripts on system startup is to use *cron* by defining *cronjobs*. As described in Listing 6.16, scripts in directory `/etc/cron.d/` have to be registered (by using *crontab*) for the user *root* to activate it as cronjobs.

Listing 6.16: Cronjobs after system boot.

```
1  # /etc/cron.d/drowsydromedary: crontab fragment
2  # for DrowsyDromedary
3  @reboot /etc/init.d/drowsydromedary start
4
5  # /etc/cron.d/sleepymongoose: crontab fragment
6  # for Sleepy.Mongoose
7  @reboot /etc/init.d/sleepymongoose start
8
9  # Activating jobs for root
10 /etc/cron.d# crontab -u root drowsydromedary
11 /etc/cron.d# crontab -u root sleepymongoose
```

6.2.6 Cloud Deployment

One possible variant of deploying N2Sky in the Clouds is depicted in the Cloud deployment diagram in Figure 6.4.

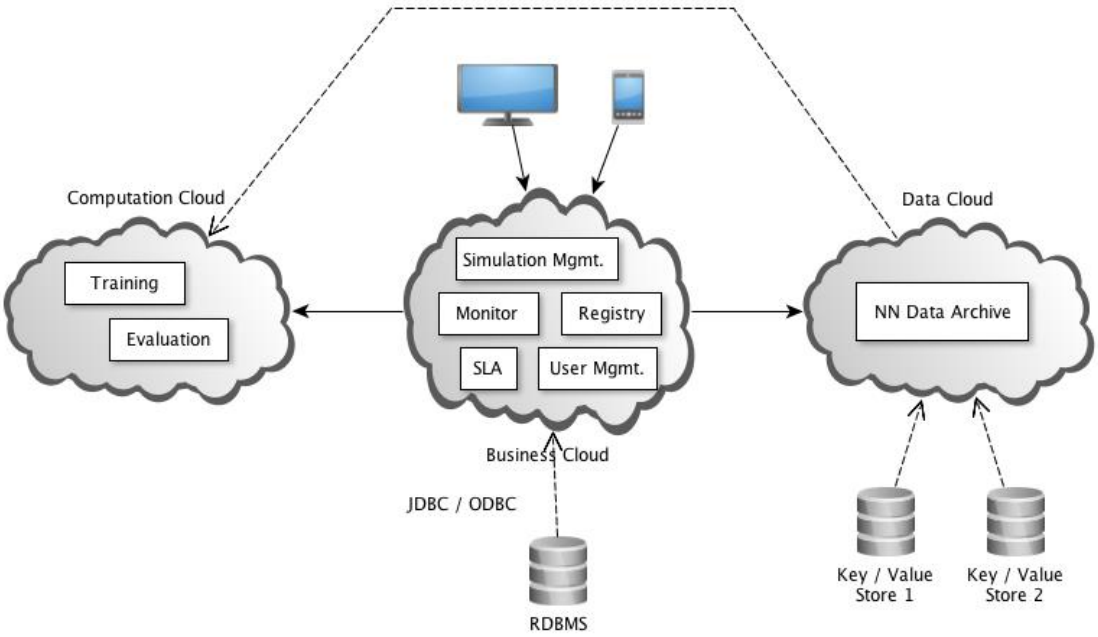

Figure 6.4: Cloud deployment diagram

Components are deployed on three different Clouds: Computation Cloud, Business Cloud and Data Cloud.

- **Computation Cloud.** The Cloud on the left hand side provides strong computing capabilities for the time-consuming training and evaluation phases of neural networks.
- **Business Cloud.** The second Cloud is the administrative Cloud, which does not provide specific hardware resources but acts as central access point for web and smart phone applications and acts as mediator to the N2Sky environment, e.g. by manageing simulation (training/evaluation) tasks or applying business models. A JDBC connection to relational database systems can be used to access our domain model.
- **Data Cloud.** The Cloud on the right hand side offers extensive storage resources for large amounts of neural network input samples and results. by key/value stores like Cassandra or document-based stores like MongoDB.

6.2.7 N2Sky Credentials

As described in Subsection 3.2.6 there are several mechanisms to ensure that only authorized stakeholders are able to access fee-based services. Four tables *(Stakeholder, StakeholderRole, RoleAssignment* and *Resource)* of the Simulation Management database (see domain model in Subsection3.3.3.2) hold up the corresponding information.

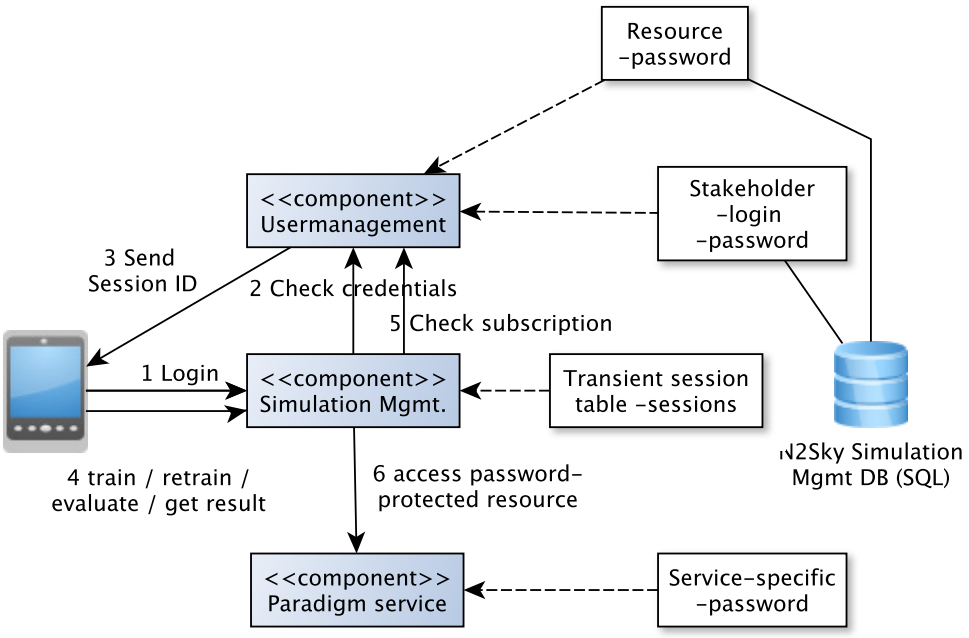

Figure 6.5: N2Sky authentication workflow.

The adapted component diagram in Figure 6.5 depicts how authentication is implemented in N2Sky and where credentials are saved:

1. After initial registration a stakeholder has login (i.e. username - 3 to 20 characters) and password (6 to 20 characters) mandatory for logging in.
2. Each client request is processed by the *Simulation Management* component, login and password is checked by the *Usermanagement* (i.e. stakeholder management) component by reading corresponding fields of the *Stakeholder* table.
3. If stakeholder authentication succeeds, a unique session id is generated and sent to the client.
4. From then on client requests add a `sessionid` parameter argument to each REST Web Service call (e.g. `train`, `retrain`, `evaluate`, `checkStatus` or `getResult`) to verify stakeholder identity.
5. If a desired paradigm service is not freely available, it has to be subscribed for a particular time period or by another pricing model. If this check succeeds, the path and password for accessing a particular service is retrieved from the *Resource* table (see Resource hierarchy in Figure 3.5).
6. With this information a password-protected paradigm service can be accessed. During initial registration of a service the service provider has to enter this password into a registration form provided during initial paradigm publishing. and store it locally so that the developed service is able to access it.

Chapter 7

Developer Guide

This chapter is divided into three sections: *Paradigm development*, *Paradigm publishing* and *SaaS Component development.*

7.1 Developing a Paradigm

To develop N2Sky-compatible paradigms basically RESTful Web Service interfaces have to be implemented consuming and producing plain text and JSON (mostly XML optionally) parameter values. If such paradigms are intended to run on an arbitrary server you are free to choose your favorite developing language to implement it. In this case you are responsible for task management (running several simulations in parallel) and security issues.

In most cases developers prefer to run their paradigm services within N2Sky's Cloud infrastructure so you are invited to use following Java classes and interfaces.

7.1.1 N2SkyService

The Java Interface *N2SkyService* stated in Listing 7.1 defines method declarations summarized in Table 7.1. *Retrain* means to use weights computed during last training as inputs for further training to achieve more exact training results. Paradigm developers have the opportunity to implement it or simply return `null`.

Every neural network paradigm service extends the abstract class *AbstractN2SkyService* implenenting *N2SkyService* (Listing 7.1) for inheriting task management for *Training* and *Evaluation* Tasks. A typical service implementation looks like the *TutorialBackpropService* in Listing 7.2.

Listing 7.1: The Java Interface *N2SkyService*.

```
1  package at.ac.univie.n2sky.common;
2  import ...
3  /**
4   * This specifies the basic methods which MUST be implemented by a
5   * N2Sky netservice.
6   */
7  public interface N2SkyService extends Remote {
8
9  /**
10  * The operation train is used to instantiate a neural network in the
11  * N2Sky service and to train this new instantiation. The operation
12  * returns a task-ID (xsd:string) for retrieving a trained neural
13  * network later. The parameters are structure, parameters and data
14  * (xsd:string valid to n2sky-*-data.xsd).
15  *
16  * @param structure valid to STRUCTURE in n2sky-definition.xsd
17  * @param parameters valid to PARAMETERS in n2sky-definition.xsd
18  * @param trainData valid to NNDATA/NNBINDATA in  n2sky-*-data.xsd
19  * @return taskID
20  * @throws java.rmi.RemoteException
21  */
22 public String train(Structure structure, Parameters parameters, N2SkyData trainData)
23    throws RemoteException;
24
25 /**
26  * The operation retrain is used to continue training on an already trained
27  * neural net service. The operation returns a task-ID (xsd:string)
28  * for retrieving a trained neural network later. The parameters are the
29  * N2Sky NetObject (see NETOBJECT in n2sky-result.xsd) and
30  * data (xsd:string valid to n2sky-*-data.xsd).
31  *
32  * @param netObject - the paradigm developer is able to create a
33  *                    paradigm-specific implementation
34  * @param trainData valid to NNDATA/NNBINDATA in  n2sky-*-data.xsd
35  * @return taskID
36  * @throws java.rmi.RemoteException
37  */
38 public String retrain(String netObject, N2SkyData trainData) throws RemoteException;
39
40 /**
41  * By the operation evaluate an already trained neural network can be sent,
42  * accompanied by evaluation data (xsd:string valid to n2sky-*-data.xsd),
43  * to this service method. The operation returns a task-ID (xsd:string)
44  * for retrieving result data later.
45  * @param netObject (part of n2sky-result.xsd) depending on paradigm developer
46  * @param evaluationData valid to n2sky-*-data.xsd
47  * @return taskID
48  * @throws java.rmi.RemoteException
49  */
50 public String evaluate(String netObject, N2SkyData evaluationData)
51    throws RemoteException;
52 }
53
54 /**
55  * The operation checkStatus provides information about a submitted training
56  * or evaluation. It returns a integer number (xsd:int) representing
57  * the progress with the following semantic:
58  *    -1 = error,
59  *    0 to 99 = percent of progress,
60  *    100 = finished.
61  * The input parameter of the operation
62  * is the task-ID (xsd:string).
63  * @param taskID
64  * @return int value of progress in percent
65  * @throws java.rmi.RemoteException
66  */
67 public int checkStatus(String taskID) throws RemoteException;
68
69 public N2SkyResult getResult(String taskID) throws RemoteException;
```

method	AbstractN2SkyService	Paradigm Service	mandatory
train		x	x
retrain		x	
evaluate		x	x
checkStatus		x	x
getResult	x		x
getResultProxy		x	x
addTask	x		
removeTask	x		

Table 7.1: Methods implemented in *AbstractN2SkyService* have to be extended in individual paradigm services.

Listing 7.2: *Tutorial Backprop Service* as an example of a paradigm service.

```
1  @Component @Path("/BPROTUTO")
2  public class TutorialBackpropService extends AbstractN2SkyService  {
3
4  @POST @Path("/train")
5  @Consumes({"application/json", "application/xml"})
6  public String trainDataMapper(N2SkyDefinition def) {
7     return train(def.getStructure(), def.getParameters(), def.getData());
8  }
9
10 public String train(
11  @HeaderParam("structure") Structure netStructure,
12  @HeaderParam("parameters") Parameters netParameters,
13  @HeaderParam("data") N2SkyData data)   {
14
15    String taskID = (new UID()).toString();
16    try {
17       N2SkyNet theNet = new TutorialBackpropNet(netStructure, netParameters);
18           this.addTask(taskID, new TrainingTask(theNet, data));
19           log.debug("train puts taskID: " + taskID + " into HashMap");
20    } catch(Exception e) {
21       this.addTask(taskID, new ErrorTask(e.getMessage()));
22    }
23    return taskID;
24 }
25
26 @GET @Path("/checkstatus")
27 public int checkStatus(@HeaderParam("taskid") String taskID)
28  throws RemoteException {                ...     }
29
30 @GET @Path("/getresult")
31 @Produces({"application/json", "application/xml"})
32 public N2SkyResult getResult(@HeaderParam("taskid") String taskID)
33  throws RemoteException {
34    N2SkyResult result = super.getResult(taskID);
35    return result;
36 }
37
38 protected N2SkyResult getResultProxy(String taskID)     { ... }
39
40 public String evaluate( ... ) { ... }
41
42 public String retrain(String netObject, N2SkyData data) {
43    // retrain not implemented
44    return null;
45 }
46 }
```

Listing 7.3: Java Interface *N2SkyNet*.

```
1  public interface N2SkyNet {
2
3  /**
4   * This Method is called in the trainThread to train the net
5   * @param data as N2SkyData (input and output values)
6   */
7  public void train(N2SkyData data);
8
9  /**
10  * This method is called from EvaluationThread to evaluate inputs
11  * @param data a N2SkyData where the input samples are
12  * represented by columns
13  */
14 public void evaluate(N2SkyData data);
15
16 /**
17  * Using this method, the tasks subscribe as listeners
18  * to the net to receive
19  * the net's status updates.
20  * @param l the StatusListener to add
21  */
22 public N2SkyNetStatus addStatusListener(StatusListener l);
23
24 /**
25  * It should return a collection of numerical samples
26  * containing matches between inputs and outputs of last evaluation.
27  *
28  * @return a collection of Samples
29  */
30 public Collection<Sample> getResults();
31
32 /**
33  * It should return a collection of binary samples (e.g. images).
34  *
35  * @return a collection of BinSamples
36 * @see at.ac.univie.n2sky.common.dto.BinSample
37 */
38 public Collection<BinSample> getBinResults();
39
40 /**
41  * This should return a diagram, if the net was trained.
42  * If evaluated it may return null.
43  * @return a training diagram
44  */
45 public Diagramm getDiagramm();
46
47 /**
48  * This method must return a string representation of the trained net.
49  */
50 public String getFlatNetObject() throws JAXBException,
51  IOException, SAXException, Exception;
52 }
```

7.1.2 Tutorial Backprop Service

TutorialBackpropService (Listing 7.2) is a demo implementation of a paradigm service annotated with Jersey [55] and JAXB binding annotations. To simplify JSON and XML posting (http method `POST`) additional training and evaluation methods are added enveloping `Structure`, `Parameters` and `NNData` to form a common `N2SkyDefinition` object.

7.1.3 N2SkyNet

Every neural network used by a N2Sky paradigm service has to implement `N2SkyNet` (Listing 7.3). A neural net typically returns either numerical data (arrays of Java `double` values) or binary files (e.g. images) therefore the interface includes both methods `getResults` and `getBinResults` whereas the net implenents only one of them and returns `null` at the other.

`GetFlatNetObject` has to return a JSON string representation of the neural network specific `NetObject` because no schema is defined for this object.

7.2 Publishing a paradigm in the N2Sky Service Store

If your paradigm has been developed and tested properly you are ready to publish your paradigm. Figure 7.1 depicts the five basic publishing steps derived from Apple's iOS app publishing process [56].

1. **Configure Paradigm Service in N2Sky's Web Portal.** Upload your paradigm as JVM-compatible `.war` file as well as the associated service description xml file valid to `n2sky-description.xml` listed in Appendix A (A.1).

2. **Submit it for approval.** After submitting the request form your paradigm service is checked automatically at first if it meets the N2Sky interface specifications described above and then is checked manually if it meets all specifications stated by the paradigm developer.

3. **Respond to approver issues.** If the service doesn't meet every specification the approval process will be paused and a mail containing approver issues is sent to the developer. After clarifying these issues the developer submits it once again by using N2Sky's web portal.

4. **Activate it** If all issues are clarified or the approved paradigm was fine from the beginning, the developer as paradigm provider is invited to activate it so his paradigm service is available for every N2Sky user.

5. **Respond to End User issues.** As part of our quality assurance strategy also end users get the opportunity to report issues. After a manual check issues are forwarded to the paradigm service provider which is again invited to respond.

Figure 7.1: Paradigm service publishing process.

The developer is invited to choose a N2Sky-wide unique service acronym for each of published paradigms. It consists of uppercase characters (paradigm group) and uppercase characters and/or numbers (paradigm):

- **Paradigm group:** The first 4 characters of service acronym. (e.g. BPRO for *Backpropagation* group).
- **Paradigm: Service acronym.** Starting from the 5th character or number.

Your service acronym is used in our databases as a further unique identifier beside a numerical primary key for better human readability and to be better prepared for reorganizaions of the resource structure. Furthermore the acronym is used to identify paradigms over Web Service endpoints.

7.3 Developing SaaS Components

7.3.1 Smartphone App - The-M-Project and jQM

We developed our web portal for mobile browsers by using pure HTML5, CSS3 and JavaScript so iOS or Android apps can be packaged via PhoneGap with minimal additional effort. We use jQuery Mobile (jQM) because it is JavaScript and CSS-based by providing a standardized way of using stylesheets and UI elements like lists, buttons or input fields. As jQM-based MVC framework *The-M-Project (TMP)* [48] is used to separate code into three categories: Model, View and Controller.

7.3.1.1 Configuration

Listing 7.4 shows a configuration code snippet in JSON style located in our project's main directory. Lines 1 to 11 show how to integrate arbitrary JavaScript libraries. Library names declared by the `name` property can be used in your JavaScript Controller code. An array of JavaScript files can be declared using a `refs` property. In our app `vkbeautify` is used to expand (pretty print) and minify input samples or results in JSON or XML format, `raphael` is a graphic library and supports visualization of

Listing 7.4: App configuration in `config.json`.

```
1  "libraries":[
2      {
3          "name": "vkbeautify",
4          "refs": ["vkbeautify.0.98.01.beta.js"]
5      },
6      {
7          "name": "raphael",
8          "refs": ["raphael-min.js",
9              "g.raphael-min.js", "g.line-min.js"]
10     }
11 ],
12 "supportedLanguages": [
13     "en_us",
14     "de_de"
15 ],
16 "defaultLanguage": "en_us",
17 "package": {
18     "myIOSConfig": {
19         "method": "PhoneGap",
20         "os": "iOS",
21         ...
22 }
```

various diagrams. Line 12 declares supported languages defined in JSON files in the `<project>/app/resources/i18n/` directory. Finally the package section defines properties for packaging a *TMP* web application into a native iOS or Android app. To create iOS apps, PhoneGap and XCode have to be installed on an OSX system. Furthermore a valid Apple developer license is needed to get a valid encryption key.

7.3.1.2 MVC Pattern

7.3.1.2.1 Model The mobile HTML5 framework *The-M-Project* implements the well-known *Model-View-Controller (MVC)* design pattern. Models, views and controllers are separated and stored in different directories. Instead of including the model part into our app (no local database) we use REST Web Service methods of our *Simulation Management* component for querying N2Sky's domain model and managing neural network training and evaluation tasks.

7.3.1.2.2 View Listing 7.5 presents a code snippet of `CurrentTrainingsView`. A `M.ListView` object defines a list of neural objects already trained or in training state ordered by training starting time. The design of a single list item is defined within a separate view file (`CurrentTrainingsTemplate`, Line 2) whereas the content of list items is linked to `CurrentTrainingsController` via `contentBinding` (lines 3 to 6). The property `idName` indicates which property will be passed if a particular `ListItem` is clicked. The remaining properties control the design of our list.

Listing 7.5: ListView definition in `CurrentTrainingsView.js`.

```
1  trainingsList: M.ListView.design({
2      listItemTemplateView: n2sky.CurrentTrainingsTemplate,
3      contentBinding: {
4          target: n2sky.CurrentTrainingsController,
5          property: 'trainings'
6      },
7      idName: 'ID',
8      hasSearchBar: YES,
9      searchBarInitialText: '',
10     isCountedList: YES,
11     isInset: NO
12 })
```

7.3.1.2.3 Controller A main task of controllers is to listen to user actions sent by Views and to send to and retrieve data from the Model. For most of our Views we created exactly one Controller for processing events initiated on its corresponding View. The code snippet in Listing 7.6 concentrates on an AJAX REST Web Service call to retrieve trained neural objects (even finished or unfinished). As we expect to receive this object list as JSON we indicate this by the property `isJSON` (line 4). Even though JSON streams are parsed automatically to JavaScript objects you have to distinguish two different cases: one training object (line 14) or a list of objects (line 9). Note that in jQuery every Web Service call is performed in an asynchronous way. This means that the programmer has to manage concurrency, the client doesn't wait for a response rather than subsequent statements are executed immediately after service call initialization. Only at the time of sending service results, one of our callback methods (`onSuccess`, `onError`) are executed depending on the received http status code. Finally, http header parameters are set in the `beforeSend` section (line 22).

Listing 7.6: AJAX Web Service call in `CurrentTrainingsController.js`.

```
1  M.Request.init({
2      url: n2sky.AdminController.get('n2s') + '/currenttrainings',
3      method: 'GET',
4      isJSON: YES,
5      onSuccess: function(data, msg, xhr) {
6          if (data != null) {
7              // More than 1 training?
8              if (data.TRAINING.length != null) {
9                  data = data.TRAINING;
10                 for (var i=0; i<data.length; i++) {
11                     data[i]['IDLABEL'] = 'ID:' + data[i]['ID'];
12                 }
13             } else {
14                 data.TRAINING.IDLABEL = 'ID: ' + data.TRAINING.ID;
15             }
16             n2sky.CurrentTrainingsController.set('trainings', data);
17         }
18     },
19     onError: function(xhr, msg) {
20         M.DialogView.alert({ ... });
21     },
22     beforeSend: function(xhr) {
23         var sid = n2sky.AdminController.get("sessionID");
24         xhr.setRequestHeader("sessionid", sid);
25     }
26 }).send();
```

7.3.2 Web Portal - Google Web Toolkit

In addition to our *Smartphone App* we plan to implement the *N2Sky Web Portal* as a unified entry point for every N2Sky stakeholder. It should be a GUI both for our *Simulation Management* and *Business Administration* components. In order to realize such portals we analyzed various Java-based frameworks (including JSF, Spring MVC/Web Flow and Oracle ADF). Due to browser heterogeneity there is no perfect framework. However we identified GWT (Google Web Toolkit) that best meets our requirements.

	The-M-Project / jQuery Mobile	Google Web Toolkit (GWT)
Development		
Included tools	TMP, jQ(M) and PhoneGap libs	GWT SDK, Plugin for Eclipse
Included server	Espresso test server	-
Requirements	Linux or OSX, Node.js >= v0.6	JDK >= v5, Eclipse >= v3.4
Design pattern	Model-View-Controller	Model-View-Presenter
Server	stateful / stateless	stateless
Client (Browser)	stateful / stateless	stateful
Develop with	Decl. JSON, JavaScript, CSS	Java, CSS
Code is generated in	HTML5, JavaScript	browser-specific JavaScript
Code gen. perform.	a few seconds (medium projects)	about 1 hour for large projects
Debugging	Browser tools	Compiler warnings, within IDE
Native iOS apps	included	via mgwt and gwt-phonegap
Deployment		
Requirements	Web server (e.g. Apache)	Application server (eg Tomcat)
Server components	not required (as app)	required (model)
Recommended for	all types except Real-Time Syst.	RCP substitution, large-scale
Licensing		
Current version	1.4.1	2.5.1
License	MIT	ASL (Apache)

Table 7.2: Differences between jQuery Mobile and Google Web Toolkit (GWT) .

The GWT DevGuide [57] provides a good overview where more detailed documentation solely is offered on Javadoc-level. GWT generally requires high learning effort in order to obtain optimum results. On the other hand both server side and client side code is Java-based, browser- and language-specific JavaScript will be automatically generated and you need not worry about specific browser representations. You have to split JavaScript files into modules otherwise the complete browser-based rich client application is loaded at once. It runs autonomously within a browser and requests JSON data via asynchronous REST Web Service calls. Thus, powerful apps can be developed that eliminate the need for time-consuming and costly launches of Rich clients. Table 7.2 compares some characteristics of GWT with those of *The-M-Project* described in previsous subsection. GWT's advanced *Model-View-Presenter* design pattern implementation allows for loosely coupled components (View doesn't contain application logic) and considerably simplified testability. By using *via mgwt* and *gwt-phonegap* it is possible to package native mobile apps so we plan to integrate N2Sky's *Smartphone App* into our GWT project.

Chapter 8

Conclusion and Outlook

This chapter points out contributions during implementation of the prototype system, it continues with a discussion of aspects not covered in this thesis although they may influence future work. The chapter ends with things I have learned during the creation of this thesis.

	Section Architect.	Section Interface	Comp. integr.	Comp. implem.
Web Portal	3.4.2	4.4.2	P	P
Smartphone App	3.4.3	4.4.3	Y	Y
Query Interface	3.4.1	4.4.1	N	N
Hosted UIs	3.4.4	4.4.4	N	N
Simulation Services	3.3.1	4.3.1	Y	Y
Simulation Management	3.3.2	4.3.2	Y	Y
Business Administration	3.3.3	4.3.3	P	P
Hosted Components	3.3.4	4.3.4	N	N
Comp. Hosting Platform	3.2.9	4.2.9	N	N
Registry	3.2.1	4.2.1	Y	Y
Monitoring	3.2.2	4.2.2	P	P
SLA	3.2.3	4.2.3	N	N
Controlling and Acclunting	3.2.4	4.2.4	N	N
Usermanagement	3.2.5	4.2.5	Y	Y
Access Control	3.2.6	4.2.6	P	P
Workflow System	3.2.7	4.2.7	N	N
Knowledge Management	3.2.8	4.2.8	N	N
Annotation Service	3.2.10	4.2.10	N	N
Component Archive	3.1.1	4.1.1	Y	Y
Data Archive	3.1.2	4.1.2	Y	Y
Ad-hoc Infrastructure	3.1.3	4.1.3	N	N

Table 8.1: Integration/implementation states of N2Sky components.

8.1 Contributions

In Table 8.1 it is stated (Yes/No/Partly) for each component if it was integrated (i.e. Web Services, APIs or protocols were applied) or implemented (i.e. the component itself was developed from the ground up or even existing components were adapted).

8.2 Conclusion and Further Research

The goal of this Master's thesis was both to provide theoretical work and a prototype system of a Cloud-based framework enabling the Computational Intelligence community to share and exchange neural network resources within a virtual organization environment. The N2Sky system provides transparent access to various resources by a layered Cloud-based architecture via defined interfaces. The N2Sky architecture was designed as an instance of RAVO allowing easy integration of resources into the Cloud services stack (SaaS, PaaS and IaaS) of service oriented architectures. The presented N2Sky prototype system has quite some room for further enhancement. Ongoing research related to N2Sky is done in the following areas:

- **Enhancing VINNSL.** We are working on an enhancement of the neural network paradigm descripion language ViNNSL [8] to allow for easier resource sharing between paradigm providers and customers. We are also aiming to build a generalized semantic description of resources for exchanging data.
- **Query interface.** A further purpose of applying semantic descriptions is to find neural network solvers for given problems, similar to a "Neural Network Google". In the course of this research we are using ontology alignment by mapping problem ontology onto neural network solution ontology.
- **Parallelize training tasks.** Parallelization of neural network training is a further key for increasing the overall performance of the N2sky system. Based on our research on neural network parallelization [58] we envision an automatically definition and usage of parallelization patterns for specific paradigms. A hot topic in this area is to select capable resources in the Cloud for execution, e.g. multi-core or cluster systems.

8.3 Lessons Learned

8.3.1 Technical Skills

- The Cloud computing paradigm: Strengths and weaknesses
- Designing and customizing service oriented architectures.
- The idea behind RESTful Web Services: How to use it the right way.
- Key/value stores: How and when they work and which kind of common query language can be used.
- Developing HTML5 GUIs (CSS3, JavaScript, jQuery Mobile, The-M-Project).
- Writing papers/thesis using LaTeX.

8.3.2 Publications Related to this Thesis

A Cloud-based Neural Network Simulation Environment

Erich Schikuta and Erwin Mann

In: *12th International Work Conference on Artificial Neural Networks.*

2013-06-12, Tenerife, Spain (2013)

N2Sky – Neural Networks as Services in the Clouds

Erich Schikuta and Erwin Mann

In: *International Joint Conference on Neural Networks,*

2013-08-04, Dallas, USA (2013)

"The nucleus accumbens is, neither pleasure nor addiction centre, and only incidentally a happiness centre. Rather, it is our brain's own learning booster. The bad news is: the module of our brain that is responsible for experiencing happiness is focused not on permanent happiness but on permanently finding interesting novelties. The good news is: those who have understood that learning and happiness are very closely linked in our minds will know that the experience of happiness is always possible throughout life. Thus answers to the question of happiness can be found precisely where you would least expect them: in learning!"

Manfred Spitzer, neuroscientist

Bibliography

[1] The Human Brain Project, "HBP website". http://www.humanbrainproject.eu/, [Last access: Sept 17, 2013].

[2] UK e-Science, "The UK e-Science Grid website". http://www.escience-grid.org.uk, [Last access: Sept 17, 2013].

[3] R. Hecht-Nielsen, "Neural network primer: Part i," *AI Expert*, pp. 4–51, Feb. 1989.

[4] R. Kruse, C. Borgelt, F. Klawonn *et al.*, *Computational Intelligence*. Springer, 2012.

[5] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations„" *International Journal of High Performance Computing Applications,*, vol. 15, no. 3, pp. 200–222, Fall 2001.

[6] R. Hilliard, "IEEE Std 1471-2000: Recommended practice for architectural description of software-intensive systems," *IEEE*, 2000, http://standards.ieee.org/findstds/standard/1471-2000.html, 2000, [Last access: Sept 17, 2013].

[7] G. Muller. A reference architecture primer. White paper, Buskerud University College, Kongsberg, Norway, 2008. Accessible at: http://www.gaudisite.nl/ReferenceArchitecturePrimerPaper.pdf, [Last access: Sept 17, 2013].

[8] P. P. Beran, E. Schikuta, T. Weishäupl, and E. Vinek, "VINNSL - Vienna Neural Network Specification Language," in *IEEE World Congress on Computational Intelligence (WCCI 2008)*. IEEE Computer Society, June 2008.

[9] K. Keahey, M. Tsugawa, A. Matsunaga, and J. Fortes, "Sky computing," *Internet Computing, IEEE*, vol. vol. 13, no. no. 5, pp. 43–51, Sept-Oct 2009.

[10] L. M. Vaquero, L. Rodero-merino, J. Caceres *et al.*, "A break in the clouds: Towards a cloud definition," *ACM SIGCOMM Computer Communication Review*, pp. 50–55, 2009.

[11] J. Geelan, "Twenty-one experts define cloud computing," 01 2009, http://cloudcomputing.sys-con.com/node/612375, [Last access: Sept 17, 2013].

[12] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," *2008 10th IEEE International Conference on High Performance Computing and Communications*, pp. 5–13, 2008.

[13] P.Mell and T.Grance, "The NIST definition of cloud computing," *Computer Security Division, National Institute of Standards and Technology, Gaithersburg, MD 20899-8930, USA, NIST Special Publication 800-145*, Sept 2011.

[14] W. Khalil, "Reference architecture for virtual organization," PhD thesis, University of Vienna, 2012.

[15] CIML, "Computational Intelligence and Machine Learning website". http://www.cimlcommunity.org/, 2011, [Last access: Dec 10, 2011].

[16] SOM Programming Team SOM-PAK, "The self-organizing map program package user guide," 1992.

[17] A. Zell, G. Mamier, M. Vogt *et al.*, "SNNS Stuttgart Neural Network Simulator user manual. Technical report," University of Stuttgart, March 1992.

[18] C. Brunner and C. Schulte, "NeuroAccess: The neural network data base system," Master's thasis, University of Vienna, 1998.

[19] E. Schikuta, "NeuroWeb: An Internet-Based Neural Network Simulator," in *14th IEEE International Conference on Tools with Artificial Intelligence*. IEEE Computer Society Press, November 2002.

[20] E. Schikuta and P. P. Beran, "A gridified artificial neural network resource," in *IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*. IEEE Computer Society, October 2007.

[21] A. A. Huqqani, L. Xin, P. P. Beran, and E. Schikuta, "N2Cloud: Cloud based neural network simulation application," in *IEEE World Congress on Computational Intelligence*. IEEE Computer Society, July 2010.

[22] IBM, "IBM Blue Cloud". http://www-03.ibm.com/press/us/en/photo/22615.wss, 2007, [Last access: Sept 17, 2013].

[23] K. Jeffery and B. Neidecker-Lutz, "The future of cloud computing, opportunities for european cloud computing beyond 2010," http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf, 2010, [Last access: Sept 17, 2013].

[24] BrainMaker, "Classify Breast Cancer Cells with BrainMaker Neural Network Software". California Scientific website, http://www.calsci.com/BreastCancer.html, [Last access: Sept 17, 2013].

[25] OGSA-DAI, "Open Grid Services Architecture Data Access and Integration (OGSA-DAI)". E-Science Grid Core Project, http://www.ogsa-dai.org.uk/, [Last access: Sept 17, 2013].

[26] B. Bonev and S. Ilieva, "Monitoring Java based SOA applications," in *Proceedings of the 13th International Conference on Computer Systems and Technologies*, ser. CompSysTech '12. New York, NY, USA: ACM, 2012, pp. 155–162.

[27] D. McCoy, "Business activity monitoring: The promise and reality," *Gartner*, July 2001.

[28] D. Parmenter, *Key Performance Indicators: Developing, Implementing,and Using Winning KPIs*. Wiley, 2011.

[29] Amazon, "Amazon Elastic Compute Cloud (EC2)". http://aws.amazon.com/ec2/, [Last access: Sept 17, 2013].

[30] OpenERP, "OpenERP website". https://www.openerp.com/de/, [Last access: Sept 17, 2013].

[31] SQL-Ledger, "SQL-Ledger website". http://www.sql-ledger.com/, [Last access: Sept 17, 2013].

[32] ADempiere, "ADempiere website". http://adempiere.org/site/, [Last access: Sept 17, 2013].

[33] LedgerSMB, "LedgerSMB website". http://ledgersmb.org/, [Last access: Sept 17, 2013].

[34] Compiere, "Compiere Community Edition website". http://www.compiere.com, [Last access: Sept 17, 2013].

[35] OpenBravo, "OpenBravo website". http://www.openbravo.com/, [Last access: Sept 17, 2013].

[36] J. Mangler, E. Schikuta, C. Witzany, J. Oliver, I. U. Haq, and H. Wanek, "Towards dynamic authentication in the grid: Secure and mobile business workflows using gset," January 2010.

[37] W. Khalil and E. Schikuta, "Virtual organization for computational intelligence," in *Human-Computer Interaction: The Agency Perspective*, ser. Studies in Computational Intelligence. Berlin, Heidelberg: Springer, 2012, vol. 396, pp. 437–464.

[38] I. Altintas, O. Barney, and E. Jaeger-frank, "Provenance collection support in the Kepler scientific workflow system," in *Proceedings of the International Provenance and Annotation Workshop (IPAW*. Springer-Verlag, 2006, pp. 118–132.

[39] L. J. Osterweil, L. A. Clarke., A. M. Ellison, R. Podorozhny, A. Wise, E. Boose, and J. Hadley, "Experience in using a process language to define scientific workflow and generate dataset provenance," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, ser. SIGSOFT '08/FSE-16. New York, NY, USA: ACM, 2008, pp. 319–329.

[40] T. Oinn, M. Greenwood, M. Addis *et al.*, "Taverna: lessons in creating a workflow environment for the life sciences: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, pp. 1067–1100, August 2006.

[41] I. Foster, J. Vöckler, M. Wilde, and Y. Zhao, "Chimera: A virtual data system for representing, querying, and automating data derivation," in *Proceedings of the 14th Conference on Scientific and Statistical Database Management*, 2002, pp. 37–46.

[42] C. Pautasso, "Composing RESTful Services with JOpera," in *Proceedings of the 8th International Conference on Software Composition*, ser. SC '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 142–159.

[43] S. Pllana, J. Qin, and T. Fahringer, "Teuta: A tool for uml based composition of scientific grid workflows," in *1st Austrian Grid Symposium. Schloss Hagenberg*. Springler Verlag.

[44] ASKALON, "ASKALON cloud and grid application development and computing environment website". http://www.dps.uibk.ac.at/projects/askalon/, [Last access: Sept 17, 2013].

[45] Sesame, "Sesame RDF website". http://www.openrdf.org/, [Last access: Sept 17, 2013].

[46] Mac Developer Library, "itunes connect Developer guide for mac developers". http://developer.apple.com/library/mac/iTunesConnectGuide, [Last access: Sept 17, 2013].

[47] Mac Developer Library, "itunes connect Developer guide for ios Developers". http://developer.apple.com/library/ios/iTunesConnectGuide, [Last access: Sept 17, 2013].

[48] The-M-Project, "The-M-Project Mobile HTML5 JavaScript Framework website". http://www.the-m-project.org/, [Last access: Sept 17, 2013].

[49] SpringSource, "Web Services API of SpringSource Hyperic HQ". http://www.hyperic.com/, [Last access: Sept 17, 2013].

[50] MongoDB, "MongoDB website". http://www.mongodb.org/, [Last access: Sept 17, 2013].

[51] Sleepy.Mongoose, "Sleepy.Mongoose website on github". https://github.com/kchodorow/sleepy.mongoose, [Last access: Sept 17, 2013].

[52] DrowsyDromedary, "DrowsyDromedary website on github". https://github.com/zuk/DrowsyDromedary, [Last access: Sept 17, 2013].

[53] Debian, "Preparation of an eucalyptus-ready cloud image," tutorial from the 2010 google summer of code project. http://wiki.debian.org/Cloud/CreateEucalyptusImage, [Last access: Sept 17, 2013].

[54] C. Bell, M. Kindahl, and L. Thalmann, *MySQL High Availability: Tools for Building Robust Data Centers*. O'Reilly, 2010.

[55] Jersey, "Jersey 2.1 user guide". https://jersey.java.net/documentation/latest/index.html, [Last access: Sept 17, 2013].

[56] Apple iOS Developer Library, "Publishing an App in the App Store," http://developer.apple.com/library/ios/#documentation/General/Conceptual/ApplicationDevelopmentOverview/DeliverYourAppontheAppStore/DeliverYourAppontheAppStore.html, [Last access: Sept 17, 2013].

[57] GWT Project, "Google Web Toolkit Developer Guide". http://www.gwtproject.org/doc/latest/DevGuide.html, [Last access: Sept 17, 2013].

[58] T. Weishäupl and E. Schikuta, "Cellular neural network parallelization rules," *CNNA '04: Proceedings of the 8th IEEE International Biannual Workshop on Cellular Neural Networks and their Applications, Los Alamitos, CA, USA, IEEE Computer Society*, 2004.

List of Figures

List of Tables

Listings

Appendix A

XML Schema Definitions

A.1 NNServiceDescription - n2sky-description.xsd

The XML Schema file with root element `NNServiceDescription` is the formal description of a paradigm service and is divided into three parts (A.1, A.2 and A.3) only in this documentation for better human readability.

Listing A.1: n2sky-description.xsd - Root element.

```
1  <?xml ...
2     <xs:include schemaLocation="n2sky-types.xsd"/>
3     <xs:element name="NNServiceDESCRIPTION">
4        <xs:complexType>
5           <xs:sequence>
6     <xs:element name="SERVICEACRONYM" type="xs:string"/>
7     <xs:element name="NAME" type="xs:string" default="ParadigmName"/>
8     <xs:element name="DESCRIPTION" type="xs:string"/>
9     <xs:element name="EXTERNALServicePATH" type="xs:anyURI" minOccurs="0"/>
10    <xs:element name="STRUCTUREDESCRIPTION">
11    ...
12    <xs:element name="PARAMETERS">
13    ...
14    <xs:element name="RETRAINABLE" type="xs:boolean"/>
15    <xs:element name="DATA">
16    ...
17          </xs:sequence>
18       </xs:complexType>
19    </xs:element>
20 </xs:schema>
```

A.2 NNDEFINITION - n2sky-definition.xsd

`NNDEFINITION` defines a new neural network as instance of a paradigm service: (Listings A.4 and A.5). *Training* inputs: `Structure, Parameters, NNData`, *Retraining* and *Evaluation* inputs: `NetObject, NNData.`

Listing A.2: n2sky-description.xsd: Structure, parameters and data.

```
<xs:element name="STRUCTUREDESCRIPTION">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="INPUTBLOCK" type="BLOCKTYPE"/>
         <xs:element name="MAXHIDDENBLOCKS" type="SIZEMAX"/>
         <xs:element name="HIDDENBLOCK" type="BLOCKTYPE" minOccurs="0"/>
         <xs:element name="OUTPUTBLOCK" type="BLOCKTYPE" minOccurs="0"/>
         <xs:element name="FULLCONNECTED" type="xs:boolean"/>
         <xs:element name="SHORTCUTS" type="xs:boolean"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>

<xs:element name="PARAMETERS">
...
<xs:element name="DATA">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="DESCRIPTION" type="xs:string" minOccurs="0"/>
         <xs:element name="MIMETYPE" type="xs:string" minOccurs="0"/>
         <xs:element name="INPUTONLY" type="xs:boolean" default="false"/>
      </xs:sequence>
      <xs:attribute name="TYPE">
         <xs:simpleType>
            <xs:restriction base="xs:string">
               <xs:pattern value="table|file"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
   </xs:complexType>
</xs:element>
```

Listing A.3: Layer block definitions.

```
<xs:complexType name="BLOCKTYPE">
   <xs:sequence>
      <xs:element default="1" name="DIMMIN" type="xs:positiveInteger"/>
      <xs:element default="1" name="DIMMAX" type="xs:positiveInteger"/>
      <xs:element default="1" name="SIZEMIN" type="xs:positiveInteger"/>
      <xs:element name="SIZEMAX" type="SIZEMAX"/>
   </xs:sequence>
   <xs:attribute name="ID" type="xs:ID"/>
</xs:complexType>
<xs:complexType name="BLOCK">
   <xs:sequence>
      <xs:element default="1" name="DIM" type="xs:positiveInteger"/>
      <xs:element name="SIZE" type="xs:positiveInteger"/>
   </xs:sequence>
   <xs:attribute name="ID" type="xs:ID"/>
</xs:complexType>
<xs:simpleType name="SIZEMAX">
   <xs:union memberTypes="xs:nonNegativeInteger">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="unbounded"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:union>
</xs:simpleType>
```

Listing A.4: n2sky-definition.xsd

```
...
<xs:include schemaLocation="n2sky-types.xsd"/>
<xs:element name="NNDEFINITION">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="SERVICEACRONYM"/>
         <xs:element name="STRUCTURE">
          ...
         <xs:element name="STRUCTUREUPLOADED" type="xs:dateTime"/>
         <xs:element name="PARAMETERS">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="PARAMETER" type="PARAMETER_TYPE"
                   minOccurs="0" maxOccurs="unbounded"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="PARAMETERSUPLOADED" type="xs:dateTime"/>
         <xs:element name="NNDATA" type="NNDATA_TYPE" minOccurs="0"/>
         <xs:element name="NNDataUPLOADED" type="xs:dateTime" minOccurs="0"/>
         <xs:element name="NETOBJECT" type="xs:anyType" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
...
```

Listing A.5: n2sky-definition.xsd - Structure definition.

```
<xs:element name="STRUCTURE">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="INPUTLayerBLOCK" type="BLOCK"/>
         <xs:element name="HIDDENLayerBLOCKS" minOccurs="0">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="HIDDENLayerBLOCK" type="BLOCK"
                   minOccurs="0" maxOccurs="unbounded"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="OUTPUTLayerBLOCK" type="BLOCK" minOccurs="0"/>
         <xs:element name="FULLCONNECTIONS" minOccurs="0">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="FULLCONNECTION" type="FULLCONNECTION"
                   minOccurs="0" maxOccurs="unbounded"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="SHORTCUTS" minOccurs="0">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="SHORTCUT" type="SHORTCUT"
                   minOccurs="0" maxOccurs="unbounded"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

A.3 NNDATA - n2sky-data.xsd

Neural network input data (Listings A.6 and A.7) is represented either as numerical (matrices of `double` values) or binary (image files) data.

Listing A.6: n2sky-data.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- N2SKY, University of Vienna,
    Faculty of Computer Science - Workflow Systems and Technology (WST) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:include schemaLocation="n2sky-types.xsd"/>
    <xs:element name="NNDATA" type="NNDATA_TYPE">
        <xs:annotation>
            <xs:documentation>
                Neural Network input data for N2SKY
            </xs:documentation>
        </xs:annotation>
    </xs:element>
</xs:
schema>
```

Listing A.7: Numerical and binary input data.

```
<xs:complexType name="NNDATA_TYPE">
    <xs:sequence>
        <xs:element name="COMMENT" minOccurs="0"/>
        <xs:element name="INPUTMATRIX" type="MATRIXTYPE"/>
        <xs:element name="OUTPUTMATRIX" type="MATRIXTYPE" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="BIN_DATA_TYPE">
    <xs:sequence>
        <xs:element minOccurs="0" name="DESCRIPTION" type="xs:string"/>
        <xs:element name="MIMETYP" type="xs:string"/>
        <xs:element name="CONTENTURI" type="xs:anyURI"/>
    </xs:sequence>
</xs:complexType>
```

A.4 NNRESULT - n2sky-result.xsd

Neural Network result - 1 result XML/JSON file per Training / Retraining / Evaluation (Listing A.8) with an included `DIAGRAM` plotting error sums per epoch (Listing A.9).

A.5 TYPES - n2sky-types.xsd

The following listings define `FULLCONNECTION` and `SHORTCUT` (Listing A.10), parameters (Listing A.12), matrices (Listing A.11) and weights (Listing A.13).

Listing A.8: n2sky-result.xsd

```
<xs:complexType>
   <xs:sequence>
      <xs:element name="SERVICEACRONYM" type="xs:string"/>
      <xs:element name="SOURCE" type="xs:anyURI">
      </xs:element>
      <xs:element name="GENERATED" type="xs:dateTime"/>
      <xs:element name="ERRORMESSAGE" type="xs:string" minOccurs="0"/>
      <xs:element name="STATUSMESSAGE" type="xs:string" minOccurs="0"/>
      <xs:element name="RESULTS" minOccurs="0">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="SAMPLE" type="SAMPLETYPE" minOccurs="0"
                  maxOccurs="unbounded"/>
            </xs:sequence>
         </xs:complexType>
      </xs:element>
      <xs:element name="BINRESULTS" minOccurs="0">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="COMMENT" type="xs:string" minOccurs="0"/>
               <xs:element name="BINRESULT" maxOccurs="unbounded">
                  <xs:complexType>
                     <xs:sequence>
                        <xs:element name="INPUT" type="BIN_DATA_TYPE"
                           minOccurs="0"/>
                        <xs:element name="OUTPUT"type="BIN_DATA_TYPE"/>
                     </xs:sequence>
                  </xs:complexType>
               </xs:element>
            </xs:sequence>
         </xs:complexType>
      </xs:element>
      <xs:element name="NETOBJECT" type="xs:anyType" minOccurs="0" />
      <xs:element name="DIAGRAM" minOccurs="0">
      ...
   </xs:sequence>
</xs:complexType>
```

Listing A.9: DIAGRAM as part of NNRESULT.

```
<xs:complexType name="SAMPLETYPE">
   <xs:sequence>
      <xs:element minOccurs="0" name="INPUT" type="MATRIXTYPE"/>
      <xs:element minOccurs="0" name="OUTPUT" type="MATRIXTYPE"/>
   </xs:sequence>
</xs:complexType>

<xs:element name="DIAGRAM" minOccurs="0">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="TITLE" type="xs:string"/>
         <xs:element name="DESCRIPTION" type="xs:string"/>
         <xs:element name="XTITLE" type="xs:string"/>
         <xs:element name="XDESCRIPTION" type="xs:string"/>
         <xs:element name="XMIN" type="xs:double" default="0"/>
         <xs:element name="XMAX" type="xs:double"/>
         <xs:element name="YTITLE" type="xs:string"/>
         <xs:element name="YDESCRIPTION" type="xs:string"/>
         <xs:element name="YMIN" type="xs:double" default="0"/>
         <xs:element name="YMAX" type="xs:double"/>
         <xs:element name="VALUES">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="PAIR" minOccurs="0" maxOccurs="unbounded">
                     <xs:complexType>
                        <xs:sequence>
                           <xs:element name="X" type="xs:double"/>
                           <xs:element name="Y" type="xs:double"/>
                        </xs:sequence>
                     </xs:complexType>
                  </xs:element>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Listing A.10: Connection definition.

```
<xs:complexType name="FULLCONNECTION">
   <xs:sequence>
      <xs:element name="FROMBLOCK" type="xs:IDREF"/>
      <xs:element name="TOBLOCK" type="xs:IDREF"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="SHORTCUT">
   <xs:sequence>
      <xs:element name="FROM">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="BLOCK" type="xs:IDREF"/>
               <xs:element name="NEURON" type="xs:integer"/>
            </xs:sequence>
         </xs:complexType>
      </xs:element>
      <xs:element name="TO">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="BLOCK" type="xs:IDREF"/>
               <xs:element name="NEURON" type="xs:integer"/>
            </xs:sequence>
         </xs:complexType>
      </xs:element>
   </xs:sequence>
</xs:complexType>
```

Listing A.11: Matrix definition.

```
<xs:complexType name="MATRIXTYPE">
   <xs:sequence>
      <xs:element name="ROW" type="MATRIX_ROW_TYPE" maxOccurs="unbounded"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="MATRIX_ROW_TYPE">
   <xs:sequence>
      <xs:element name="C" type="xs:double" maxOccurs="unbounded"/>
   </xs:sequence>
</xs:complexType>
```

Listing A.12: Parameter definition.

```
<xs:complexType name="PARAMETER_TYPE">
   <xs:sequence>
      <xs:element name="NAME" type="xs:string"/>
      <xs:element name="DESCRIPTION" type="xs:string" minOccurs="0"/>
      <xs:element name="VALUE" type="xs:string"/>
      <xs:element name="DEFAULTVALUE" type="xs:string" minOccurs="0"/>
      <xs:element name="COMBOOPTIONS" type="COMBOOPTIONS_TYPE" minOccurs="0"/>
   </xs:sequence>
   <xs:attributeGroup ref="PARAMETER_ATTRIBUTES"/>
</xs:complexType>
<xs:attributeGroup name="PARAMETER_ATTRIBUTES">
   <xs:attribute name="ID" type="xs:ID"/>
   <xs:attribute name="TYPE">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:pattern value="value|bool|combo"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
</xs:attributeGroup>
<xs:complexType name="COMBOOPTIONS_TYPE">
   <xs:sequence>
      <xs:element name="OPTION" type="xs:string" maxOccurs="unbounded"/>
   </xs:sequence>
</xs:complexType>
```

Listing A.13: Definition of weights between neurons.

```
<xs:complexType name="WEIGHTS">
   <xs:sequence>
      <xs:element name="FROMBLOCK" type="xs:IDREF"/>
      <xs:element name="TOBLOCK" type="xs:IDREF"/>
      <xs:element maxOccurs="unbounded" name="FROMNEURON">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="NUMBER" type="xs:integer"/>
               <xs:element name="TONEURON">
                  <xs:complexType>
                     <xs:sequence>
                        <xs:element name="NUMBER" type="xs:integer"/>
                        <xs:element name="WEIGHT" type="xs:float"/>
                     </xs:sequence>
                  </xs:complexType>
               </xs:element>
            </xs:sequence>
         </xs:complexType>
      </xs:element>
   </xs:sequence>
</xs:complexType>
```

Appendix B

Abstract

B.1 English

Working with artificial neural networks typically is very time-consuming and "resource-hungry". Neural network training and evaluation tasks cause the main part of the effort. This problem leads to the basic idea of the present Master's thesis, namely to create a neural network simulation environment consisting of services in the Clouds. Using a service-oriented architecture allows for a flexible and reusable component structure and Cloud computing ensures elasticity in terms of computing power and available resources. Sky computing is about joining different Clouds to create large infrastructures.

N2Sky is a novel framework implementing these concepts. It provides methods for processing and exchanging neural network-specific artifacts such as neural paradigms and objects within a virtual organization environment. Transparent access to various resources is provided by a layered Cloud-based architecture via defined interfaces. This allows a large group of users to make use of neural networks as simple as possible - from beginners to experienced researchers. The reference architecture for virtual organizations (RAVO) allows easy integration of resources into the Cloud services stack (SaaS, PaaS and IaaS) of service oriented architectures.

B.2 Deutsch

Das Arbeiten mit künstlichen neuronalen Netzen ist in der Praxis meist sehr zeit- und ressourcenintensiv. Das Trainieren und Evaluieren von neuronalen Netzen macht dabei den Großteil des Aufwands aus. Diese Problemstellung führt zur Grundidee der vorliegenden Masterarbeit, nämlich eine service-basierte Simulationsumgebung für neuronale Netze in Clouds zu entwickeln. Die Verwendung einer service-orientierten Architektur ermöglicht eine flexible und wiederverwendbare Komponentenstruktur und Cloud Computing sorgt für Elastizität hinsichtlich Rechenleistung und verfügbaren Ressourcen. Sky Computing bedeutet darüber hinausgehend die Zusammenführung unterschiedlicher Clouds zu großen Infrastrukturen.

N2Sky ist ein neu entwickeltes Framework, das diese Konzepte umsetzt. Es stellt Methoden für die Verarbeitung und den Austausch von netzspezifischen Artefakten wie neuronalen Paradigmen und Objekten innerhalb einer virtuellen Organisation bereit. Die in Schichten unterteilte Cloud-Architektur erlaubt den transparenten Zugriff auf unterschiedlichste Ressourcen über definierte Schnittstellen. Damit soll die Verwendung neuronaler Netze einem großen Nutzerkreis - vom Einsteiger bis zum erfahrenen Forscher - so einfach wie möglich gemacht werden. Die Referenzarchitektur für virtuelle Organisationen (RAVO) ermöglicht die einfache Integration von Ressourcen in den Cloud-Service-Stack (SaaS, PaaS sowie IaaS) von service-orientierten Architekturen.

Appendix C

Curriculum Vitae

Erwin Mann

Geboren in: Hollabrunn, Niederösterreich

Ausbildung

Bachelorstudium Wirtschaftsinformatik	2006-2010
Masterstudium Wirtschaftsinformatik	2010-2013

Publikationen

A Cloud-based Neural Network Simulation Environment
Erich Schikuta and Erwin Mann
In: *12th International Work-Conference on Artificial Neural Networks*
2013-06-12, Tenerife, Spain (2013)

N2Sky – Neural Networks as Services in the Clouds
Erich Schikuta and Erwin Mann
In: *International Joint Conference on Neural Networks*
2013-08-04, Dallas, USA (2013)