Service provision in a utility computing environment

B.Heckmann

Network Research Group, University of Plymouth, Plymouth, United Kingdom
email: benjamin.heckmann@gmx.de

Abstract

This project is motivated by the gap between technology-centred service provisioning frameworks and the business model Utility Computing. In the beginning this paper introduces the term 'Utility Computing' [UC] as an on-demand service provision business model for Service-oriented Architectures. It distinguishes Utility Computing from technology-originated terms such as Grid or J2EE.

The paper describes these technologies as possible frameworks to implement IT architectures for UC business models. And it determines that actual frameworks are not smart enough to fit the service provisioning demands of small to medium-sized businesses. Therefore a technology-independent and UC-conform service provisioning model is claimed, that enables framework evaluations and simulations of provisioning demands.

Subsequently, the basic structure for a technology-independent, UC-conform service provisioning model is described. As a first step towards such a model this paper introduces the general conditions for such a network, underlying use cases, derived network elements and appropriate workflows. With this as base the overall project aims to provide a technology-abstracted model for service provision and fundamentals for load characteristic simulations for UC environments.

Keywords

SaaS, On Demand, Utility Computing, Grid, Service-oriented Architecture, Service Billing, Web Service Provision, Quality of Service

1. Introduction

1.1. On Demand Service Provision for Service-oriented Architectures

Service-oriented Architectures [SOA] (MacKenzie *et al.*, 2006) are one of the most observed topics in IT today. More and more standard software products are delivered 'SOA-ready', which in most cases means additionally equipped with a webservice interface. This paper will focus on SOAP-based webservices (Booth *et al.*, 2004) as implementation technology for SOAs.

On the service consumer side 'SOA-readiness' means that the encapsulated functionality becomes accessible to each business process step separately. As a result, you can easily rearrange your business processes, while your backend software stays untouched.

On the service provisioning side this means that with standardised interfaces and firewall-friendly protocols, service provision could evolve to its next step: from locally deployed purchased software packages to remotely hosted pay-per-use services.

This business model of leasing remote services and charging the customers per usage is defined with the term 'Utility Computing' [UC] in this paper. It is also known under the terms Software-as-a-Service [SaaS] or On-Demand Computing. Additionally, this paper discusses UC frameworks, networks, models and further elements that can act as part of the implementation of a Utility Computing business model.

1.2. Utility Computing frameworks for small and medium-sized businesses

Future works should be concerned with the question of whether there a suitable UC framework for the service provision in small and medium-sized businesses [SMB] based on open source software. Currently the estimated answer is: not yet.

To be able to latter follow up this question, an abstract model for a UC framework is required. This paper will attempt to define the basic elements and workflows for a UC network in preparation for designing a technology-abstracted UC model.

1.3. Meshed services and resource prediction

This project also aims to reach conclusions about basic questions linked with the operations of UC services. Simulations of service network behaviour, based upon the model to be elaborated, should shed light on the following questions:

- The behaviour of *highly meshed UC services* in response to service failures and in case of service loops.
- The ability for *resource prediction* for UC service providers to identify bottlenecks in the UC infrastructures including service clients, the UC service network and embedded foreign services.
  Planning of peak demand scenarios for provided services and basic operating figures for pricing strategies, which should be covered by information gathered for bottleneck identification in conjunction with variations of the simulated scenarios.

To recapitulate, this project should deliver a technology-independent, UC-conform model as a background for future UC framework evaluations and simulations of UC provisioning scenarios.

1.4. Outline of research

In the following second chapter the conceptual approach for the overall project is described. The chapter describes the three major steps towards a UC model that enables evaluations of technology-dependent UC implementation frameworks: context gathering, model building and confirmation of the model.

Afterwards the terms Utility Computing, Grid and J2EE are demarcated in the third chapter. The demarcation provides a better understanding for the coherence between UC as a business model and technology frameworks like Gird or J2EE.

As core subject of this paper the basic elements and workflows of a UC network are introduced in chapter four. The elements and corresponding basic workflows are based on a service consumption and a service provision use case. These use cases are derived from works from OGSA, GGF and industry best practices like ITIL.

2. Conceptual approach for the project

2.1. Pre-modelling context building

In the context phase, the project defines its basic terms and elaborates its background and related work. The most important term should be Utility Computing itself. As a business model it is technology-independent and focused on economic opportunities of the utility idea.

As a result, detailed analyses of provisioning costs or capacity demands pre to investments can not be made. Also, comparisons of different provision technologies are not possible. This is due to a missing technology-abstracted service provisioning model substantiating the business model.

In addition, possible current technologies to provide UC-conform services should be examined and demarcated. The gathered information should provide a basis for provisioning conditions.

2.2. Technology-independent, UC-conform service provisioning model

The model building phase of the project is separated into four consecutive steps:

(1) As an initial point for model building, the use cases collected in the context of the Open Grid Services Architecture [OGSA] and results of the EU GRASP project that aim to define an infrastructure for Application Service Provision [ASP] based on GRID technology will be reused.
Additionally, the model should consider the basic characteristics of industry standards like ITIL or CobiT. Based on these existing use cases and industry-class service delivery demands, new UC-centred uses cases must be derived.

(2) Taking the derived UC-centred use cases as a basis, the technology-independent and UC-conform service provisioning components must be identified.

(3) The basic workflows for the component interaction must be described. They should at least enable the model to provide services that are scalable over cost-domains and can be billed according to customer usage.

(4) A complete model must be built based on the defined service provisioning components and workflows. This model should be usable as a basis for simulation-based analyses of UC networks.

2.3. Simulation-based analyses

In the simulation phase of the project, the previously developed model will be utilised for the implementation of a simulation environment for UC-conform service provisioning.

The following activities are necessary:

(1) A suitable simulation framework for the developed model must be selected.
(2) The model must be implemented within the selected framework.
(3) First simulations should be accomplished. They should address the behaviour of highly- meshed UC services and provide the basis for the examination of peak demands within the simulated model.

3. Background and Related Work

As preparation for defining the Utility Computing model, this paper defines and demarcates the terms Utility Computing, Grid and J2EE.

3.1. UC, Grid and J2EE definition

Utility Computing

Utility Computing describes a business model to offer software-based services in the future. While today we are becoming increasingly reliant on computer technology, an interesting question arises: "Is computing the next utility?" (Rappa, 2004)

To answer this question, the term 'utility' first should be defined. The difference in offering a 'service' to a customer or a customer who utilises a 'utility' is shaped by the underlying requirements on the consumer side: necessity, reliability, usability, utilisation, scalability and exclusivity. Additionally, the business model is based on the metering of usage combined with a 'pay as you go' approach. For more detailed description see (Rappa, 2004).

From the service consumer perspective, the most important advantages of Utility Computing are "the reduction of IT-related operational costs and complexity" (Shin Yeo *et al.*, 2006). The investments for the IT infrastructure are no longer static costs for technology and operating staff, but now depend on the usage of the utilised services. As a result the costs become variable.

On the other hand service providers can serve their resources to a wide spread number of users with diverse usage patterns. This increases the chance to minimise unutilised resources on the provider side. "Utility computing also enables providers to achieve a better Return On Investment (ROI) such as Total Cost of Ownership (TCO) […] ." (Shin Yeo *et al.*, 2006) For more detailed description see (Shin Yeo *et al.*, 2006).

Due to the 'pay-per-use' approach of UC there is a new direct relation between IT service provisioning costs and business process costs, especially in the context of Service Oriented Computing [SOC] (Munindar and Huhns, 2005). Following this approach, the costs for processes that utilise UC-based services are quite easy to comprehend. "The provider may be an organization's IT department or an external utility provider, and the service may be storage, computing, or an application." (Foster and Tuecke, 2005)

Grid

Basically, a Grid "coordinates resources that are not subject to centralized control" (Foster, 2002). This means that it is a system that is able to dispose requests for a certain functionality under known resources, regardless of the administrative domain in which a resource is hosted.

One major aspect for achieving this ability is "using standard, open, general-purpose protocols and interfaces" (Foster, 2002) to build a Grid. The final needed characteristic of a Grid is that it must be able "to deliver nontrivial qualities of service" (Foster, 2002), which implies that in a Grid "the utility of the combined system is significantly greater than that of the sum of its parts" (Foster, 2002). For more detailed description see (Foster, 2002) and (Foster and Kesselmann, 2004).

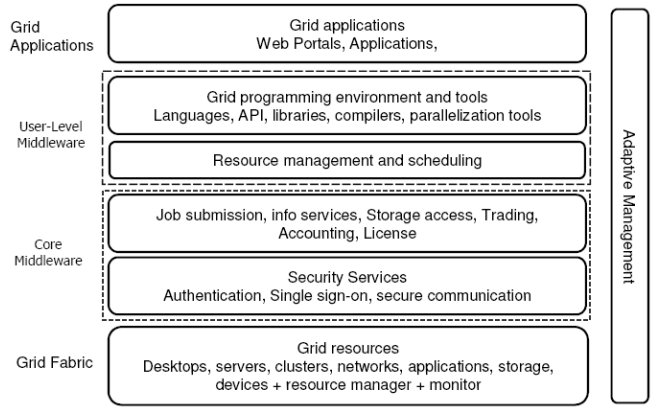

Figure 1: Grid layers (Shin Yeo *et al.*, 2006)

J2EE

J2EE is an application model that supports applications that implement enterprise services. "Such applications are inherently complex, potentially accessing data from a variety of sources and distributing applications to a variety of clients." (Sun, 1999) The middle tier of this application model offers its deployed services to consumers. It handles properties like high availability, security and scalability, "to insure that business transactions are accurately and promptly processed" (Sun, 1999). To store the data processed by the middle tier services the EIS-Tier is used. For more detailed description see (Sun, 1999).

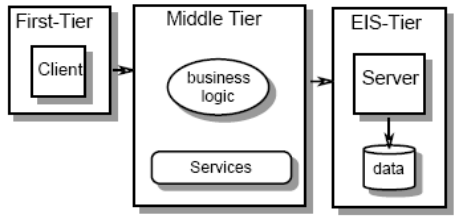

Figure 2: J2EE architecture (Sun, 1999)

3.2. UC, Grid and J2EE demarcation

UC vs. Grid

Utility Computing as a business model requires a technical environment to offer its services. Grids have the potential to serve as an appropriate service host. Grids aim to enable resource sharing and problem solving on an infinite number of computing devices. As a result, multi-institutional virtual organizations can be built upon a wide range of computing devices that are logically coupled together and presented as a single unified resource. "The design aims and benefits of Grids are analogous to those of utility computing, thus highlighting the potential and suitability of Grids to be used as utility computing environments." (Shin Yeo *et al.*, 2006) For more detailed description see (Shin Yeo *et al.*, 2006).

Grid vs. J2EE

To implement Grid services, a specific hosting or execution environment is needed. This environment is characterised through certain development tools and programming languages that meet the Grid service semantics. Previous Grid applications are realised by relying on native operating system processes as their hosting environment.

Modern container- or component-based hosting environments such as J2EE can also be used to implement Grid services. These environments offer a framework to build complex applications that offers superior programmability, manageability, flexibility and safety. For more detailed description see (Foster *et al.*, 2002).

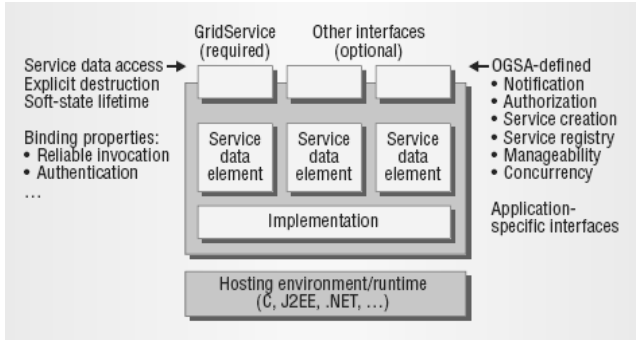

Figure 3: Grid architecture (Foster *et al.*, 2002)

UC vs. J2EE
The common trend as described in 'Grid vs. J2EE' is using a Grid that is J2EE-based. For an example of building a J2EE-based Grid, see (Araki, 2004).

A standalone solution for J2EE-based UC is not known. J2EE-clusters are possible, but without billing and cross-side (and therefore cross-cost zones) load-balancing.

Cluster definition: "group of machines working together to transparently provide enterprise services" (Kang, 2001)

3.3. Demarcation summary

Utility Computing can best be described as a business model for offering services within or to organisations. A Grid could be one technology to build and offer UC-based services. With Grid environments, however, "there is a fundamental gap between the technology and its users" (De Roure *et al.*, 2006). The targeted audience in this project are SMB. For this audience the technology is still too complex and requires too much knowledge commonly not available in-house. For more detailed description see (De Roure *et al.*, 2006).

J2EE as standalone technology is not able to offer UC-based services. Solutions for service-consumer billing or cross-side (cross-cost-domain) load-balancing are lacking within J2EE.

4. Definition of the basic elements of a UC network

4.1. Modelling properties and use cases

The goal for the model building is to at least fulfil the minimum requirements of Utility Computing, which are service provision 'on-demand' and 'pay-per-use' billing. The targeted properties are (currently excepted is the service transaction management):

- SOA service provision
- Extensive load-balancing
- Management of service quality
- Accounting
- Model complexity fitting for SMB (service provision and consumption side)

4.2. Underlying use cases and general conditions

The analyses of the functional requirements are based on the OGSA and GGF use cases (Foster *et al.*, 2004) (MacLaren *et al.*, 2006) (Von Reich, 2004):

- Commercial Data Centre
- Grid Resource Resellers
- Inter Grid
- Resource Usage Service

- IT Infrastructure and Management
- Grid-based ASP for Business
- Grid Monitoring Architecture

Additionally, it is based on the main results of the EU GRASP project that aims to define an infrastructure for Application Service Provision based on Grid technology (Dimitrakos *et al.*, 2004).

Complementing the basic requirements in the industry standard ITIL with focus on service delivery best practices are incorporated. Also basic requirements from the CobiT (ISACA, 2005) framework are included.

4.3. Derived use cases for the model

Starting from the underlying use cases and general conditions brought together previously, the following two use cases define the basic functional requirements the model should fulfil. Aggregating the service delivery requirements and matching them against the predefined goals for the model resulted in the subsequently-denoted use cases for UC service delivery operation.

Service consumption use case overview

- Discovery
- Brokering and load-balancing
- Orchestration
- Authentication and Authorisation
- Monitoring, Metering and Accounting
- Fault Handling and Logging
- Corresponding Policies

Service provision use case overview

- Data Access
- Provisioning
- Embedded legacy applications
- Synchronous and asynchronous usage
- Administration
- Corresponding Policies

4.4. Elements derived from the model use cases

- Service type
  The element represents a definition of a service class with distinctive business functionality and a standardised public interface.
- Service instance

This represents an instance of a service type that can handle multiple service requests simultaneously, and exists as a subset of a service host and applies SLA quotas. The element supports standby, online and offline modes.

- Service host
  The element represents a host for service instances that can only host one service instance of a service type at a time.
- Service consumer
  The consumer invokes service instances by sending service requests.
- Service request
  A request is an invocation of a service initiated by a service consumer. The invocation always includes the associated service response (synchronous or asynchronous).
- Service registry
  The registry authenticates service consumers.
- Service broker
  A broker authorises service requests, forwards service requests to the most suitable service load-balancer or third-party service broker (with respect to SLA and cost calculations) and creates service request bills (including third-party service type utilisation costs and SLA violations).
- Service load-balancer
  The load-balancer represents a physical location or a cost class. It queues service requests (if necessary) and forwards service requests to most suitable service instances (with respect to SLA). Also it deploys, activates, deactivates or removes service instances on service hosts as necessary (e.g. for load-balancing or in case of failure).
- Service monitoring
  This element monitors the SLAs per service request.
- Policies
  These elements define the general conditions for brokering (per service consumer), error and event handling (per service type) and additionally security conditions (per service consumer).
  Information is stored near their creation or consumption location. Information is provided directly through its storage location.
- Variations to the derived use cases for the model
  Not incorporated in the element definition are the embedded legacy applications, administration and policies areas of the service provision use case.
  The orchestration of existing services into new services is indirectly supported through the provision of new service types. This means that if you want to orchestrate existing service types to new service types, you must build a new service type and as internal functionality invoke and compose the existing services.

4.5. Workflows for the model

In workflow steps the '→' sign is read as 'requests' and marks a request track. Steps can be marked as optional to the initially requesting instance. Steps marked as

'*TERMINATOR*' are always executed at the end of any workflow, regardless of the workflow type (e.g. 1_SSC, 2_CoSC, 3_CaSC).

1_SSC: Simple service consumption workflow

The following workflow describes the simplest possible service request in the model:

1) Service consumer → Service registry *OPTIONAL*
   Authenticated service consumers can request service type information including cost information and available service brokers.
2) Service consumer → Service broker → Service load-balancer → Service instance
   Authenticated service consumers can send service requests using a service broker. In response they get the service state and a request bill.
3) Service broker → Service registry
   This step is invoked by step 2 and transmits the service consumer authentication data and service type to get authorisation information and brokering policy.
4) Service load-balancer → Service monitoring → Service host
   This step is invoked by step 2 and collects the service hosts load data.
5) Service instance → Service monitoring
   This step is invoked by step 2 and reports the individual service request load during processing and according events.
6) Service broker → Service monitoring *TERMINATOR*
   This step is invoked by step 2 and closes a service request by reporting third-party service usage information and the issued service bill to the monitoring service.

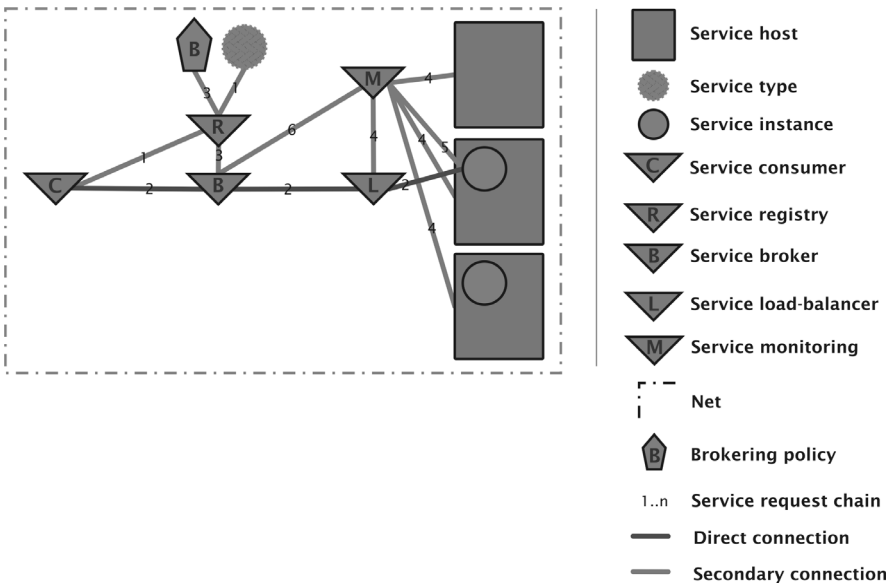

Figure 4: 1_SSC network view

2_CoSC: Complex service consumption workflow

The workflow for complex service consumption expands the basic workflow for simple service consumption [1_SSC]. It describes a more complex workflow within the model by still providing a single service type.

7) Service broker → Service load-balancer
   This step is invoked by step 2 and collects the service type utilisation information per load-balancer.

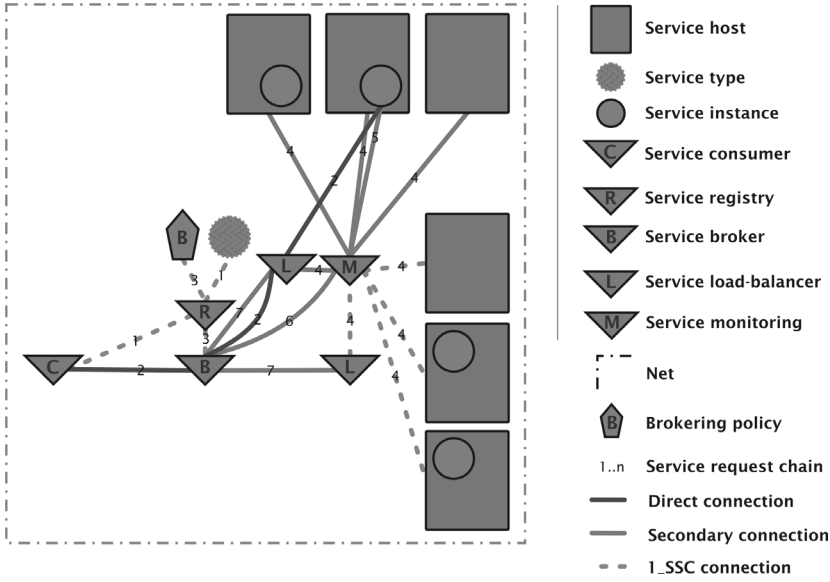

Figure 5: 2_CoSC network view

3_CaSC: Cascaded service consumption workflow

The workflow for cascaded service consumption expands the workflow for complex service consumption [2_CoSC]. It describes a workflow that utilises an externally-provided service.

8) Service instance (local) → Service broker (local) → Service broker (third-party) → Service load-balancer (third-party) → Service instance (third-party)
   This step is invoked by step 2 and invokes a third-party service. The invoking instance sends its service request enhanced with service consumer authentication data by the service broker. In the responding data, the service request response and state are used by the service instance. The request bill is extracted by the service broker.
9) Service broker (local) → Service broker (third-party) → Service monitoring (third-party)
   This step is invoked by step 8 and collects the service type utilisation information for the third-party service.

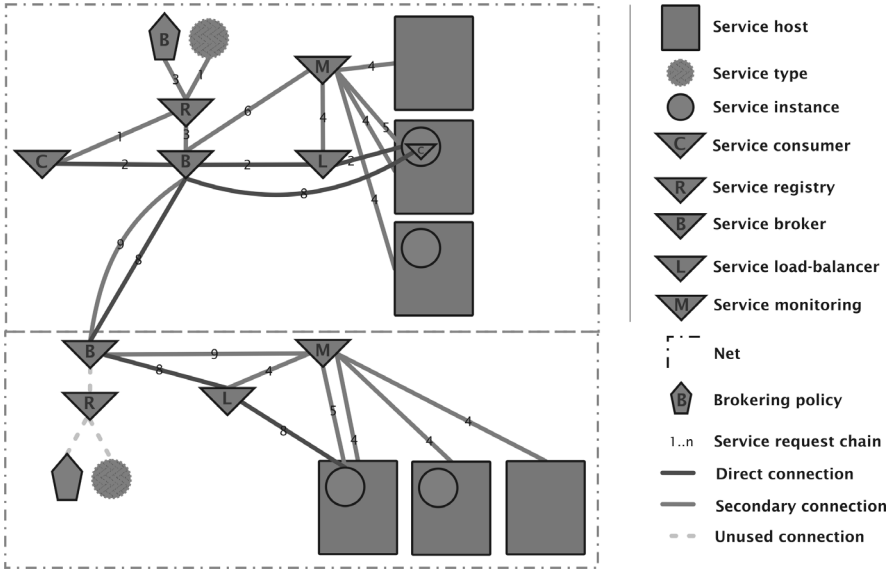

Figure 6: 3_CaSC network view

Service instances as service consumers

Why service instances should not invoke their embedded service calls directly: Service instances need to act as service consumers, when they want to embed functionality provided by other services. If service instances would call their embedded services directly, the system would lose control over:

- Service provider changes
- Authentication changes
- Service billing
- Service load-balancing

With a centralised element as provided with the service broker, external service invocation will be handled by the broker. This introduces a new implementation strategy for software developers of service-oriented architectures.

4.6. Enhancements compared to plain SOA

The four main differences to service provision in basic service-oriented architectures are:

- *Pay-per-use base*, achieved through the optional usage of the service request bill
- *Internal active SLA-control*, achieved through the service type utilisation combined with the service load-balancer
- *External passive SLA-control*, optionally achieved through the service type utilisation combined with the service broker

- *Centralised service consumption management*, achieved through the service broker
  Thus not only is the service provision managed through a central proxy-like instance, but the service consumption is also managed centrally.

As an analogy, compare the evolved architecture with the IBM proposal for a utility computing architecture in (Kloppmann *et al.*, 2004)..

5. Summary

This paper introduced Utility Computing as the on-demand service provision business model for Service-oriented Architectures. As a trigger for the project, the question for a suitable UC framework for small and medium-sized businesses was raised. Subsequently, a technology-independent, UC-conform service provisioning model was claimed as a precondition to answer this question.

The paper goes on to expresses the need to characterise the behaviour of meshed services and the necessity to provide room for resource prediction in UC networks. As a suitable solution, a simulation framework based on the previously demanded UC model is described.

To distinguish Utility Computing clearly from technologies such as Grid or J2EE these terms are defined and demarcated. The conceptual approach for the project is explained and the first steps towards a technology-abstracted UC model are presented. The gathering of the basic network elements including the general conditions for the network, its developed uses cases, the derived network elements and the appropriate workflows are introduced.

Based on these results, the project will now start to build a complete UC model as a precondition for the simulation phase of the project.

6. References

Araki, T. 2004. *Autonomic WWW server management with distributed resources*. In Proceedings of the 2nd Workshop on Middleware For Grid Computing (Toronto, Ontario, Canada, October 18 - 22, 2004). MGC '04, vol. 76. ACM Press, New York, NY, 81-86.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D., 2004. *Web Services Architecture*. W3C Working Group Note 11, http://www.w3.org/TR/ws-arch/.

Chair: K. Jeffery; Editor-in-Chief: D. De Roure, 2006. *Future for European Grids: GRIDs and Service Oriented Knowledge Utilities*. European Commission, published in January 2006.

Chee Shin Yeo, Marcos Dias de Assunção, Jia Yu, Anthony Sulistio, Srikumar Venugopal, Martin Placek, and Rajkumar Buyya, *Utility Computing on Global Grids*, Hossein Bidgoli (ed), The Handbook of Computer Networks, John Wiley & Sons, New York, USA, accepted in April 2006 and in print.

Dimitrakos, T., Mac Randal, D., Wesner, S., Serhan, B., Ritrovato, P., Laria, G., 2004. *Overview of an architecture enabling Grid based Application Service Provision*. AxGrid '04 , Nicosia, 28-30 January, 2004.

Foster, I., 2002. *What is the Grid? A Three Point Checklist*. Argonne National Laboratory & University of Chicago.

Foster, I., Gannon, D., Kishimoto, H., von Reich, J.J., 2004. *Open Grid Services Architecture Use Cases*. GGF, http://www.ggf.org/documents/GFD.29.pdf.

Foster, I., Kesselmann, C., 2004. *The Grid 2 – Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers.

Foster, I. Kesselman, C. Nick, J.M. Tuecke, S., 2002. *Grid services for distributed system integration.* Computer, Volume 35, Issue 6, June 2002 Page(s):37 – 46

Foster, I. and Tuecke, S. 2005. *Describing the elephant: the different faces of IT as service*. Queue 3, 6 (Jul. 2005), 26-29.

ISACA, 2005. *COBIT 4.0*. Printed in the United States of America, 2005. ISBN 1-933284-37-4

MacLaren, J., Newhouse, S., Haupt, T., Keahey, K., Lee, W., 2006. *Grid Economy Use Cases*. GGF, http://www.ggf.org/documents/GFD.60.pdf.

Jeffrin J. Von Reich, 2004. *Open Grid Services Architecture: Second Tier Use Cases*. OGSA-WG GGF, http://forge.gridforum.org/sf/docman/do/downloadDocument/projects.ogsa-wg/docman.root.published_documents.use_cases_1_0/doc13574.

Kang, A., 2001. J2EE clustering, Part 1. JavaWorld.com

Kloppmann, M., Konig, D., Leymann, F., Pfau, G., Roller, D., 2004. *Business process choreography in WebSphere: Combining the power of BPEL and J2EE*. IBM Systems Journal, Volume 43 Issue 2.

MacKenzie, C.M., Laskey, K., McCabe, F., Brown, P.F., Metz, R., Hamilton, B.A., 2006. *Reference Model for Service Oriented Architecture 1.0*. OASIS Committee Specification 1, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.

Munindar P. Singh, Michael, N. Huhns, 2005. *Service Oriented Computing Semantics, Process, Agents*. Hrsg: John Wiley & Sons.

Rappa, M. A. 2004. *The utility business model and the future of computing services*. IBM Syst. J. 43, 1 (Jan. 2004), 32-42.

Simplified Guide to J2EE, 1999. Sun Microsystems, Inc.