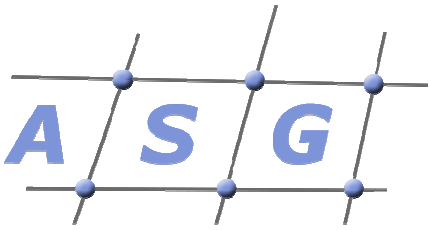
Fraunhofer Institut
Experimentelles
Software Engineering

Requirements Specification Survey

Adaptive Services Grid Deliverable D6.I-1



Authors:
Michael Eisenbarth
Tom Koenig
Thomas Olsson

Funded and supported by the European Union Project ASG (FP6-IST-004617).

IESE-Report No. 133.05/E
Version 1.0
February 28, 2005

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competitive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach (Executive Director)
Prof. Dr. Peter Liggesmeyer (Director)
Fraunhofer-Platz 1
67663 Kaiserslautern

Executive Summary

The goal of Adaptive Services Grid (ASG) is to develop a proof-of-concept prototype of an open development platform for adaptive services discovery, creation, composition, and enactment. To achieve its goal, ASG addresses scientific and technological issues especially related to the field of software engineering.

Requirements engineering is one of the key areas in successful software engineering and software development. Requirements engineering deals with system requirements that are specifications of the services a system should provide, the constraints on the system and additional background information, which is required to develop the system. Requirements engineering is the systematic process concerned with the elicitation, understanding, analysis and documentation of the system requirements. Requirements engineering itself is called an "engineering" process as the process itself is used in practical and systematic way where trade-offs have to be made to find the best solution. An important aspect of requirement trade-off that must be addressed within the ASG development methodology is the relationship between service customers and service providers. Based on semantic specifications of requested services by service customers and the semantic specifications of the registered services by service providers, ASG has to discover appropriate services, composes complex processes and – if required – generates software to create new application services on demand. Consequently, application services will be provided through the underlying computational grid infrastructure based on adaptive process enactment technology.

Requirements seem to be prominent especially on three different levels of the ASG platform. The first level is the service customer level that represents the level of the ASG platform at which service requests are raised. The users of the Adaptive Services Grid thus request a desired functionality based on needs for fulfilling a customer related business process. The second level is the service grid level, which represents the level of the ASG platform at which the service discovery and composition takes place based on the raised service request by a customer. On this level, the ASG platform determines which existing services can fulfil the customer requests or if new services have to be generated. The third level that has to be addressed is the service provider level that represents the level of the ASG platform at which service providers semantically specify their provided services and register them to the service registry of the ASG platform so that a service can be discovered and executed to fulfil a customer request. The service specification must also include quality of service specifications to express constraints on the execution of services and to identify services that are capable of fulfilling a customer request within resource boundaries.

This report describes the current state-of-the-art and state-of-the practice of requirements specification techniques and a framework on how these techniques can be applied for system requirements engineering in a service-oriented environment. The survey is thereby focussed on the relationship between use case based specification techniques, services, and business processes that aim at understanding how services can be used to determine processes and use cases (i.e., service-based application specification) and how these techniques can be applied to the three different levels of requirements mentioned above.

The survey describes specifications methods and notations used for functional and non-functional (QoS) service requirements. Another important aspect addressed in this report will be the introduction of ontology-based information modeling. Ontologies are formal models of a specific domain that support the communication between human and computer based actors. This facilitates exchange and sharing of knowledge within an organization. Especially in a distributed service-oriented environment as the ASG platform, various partners or organizations would like to provide and share data or other resources to customers, for example service specifications or code components. This survey will be used as a basis to develop a requirements engineering method for the ASG development method (ASG project deliverable D6.I-2).

Keywords: Service-oriented requirements engineering, service-based application specifications, Quality of services, Ontology-based information modeling, ASG

Table of Contents

1 Introduction

This report describes the current state-of-the-art and state-of-the practice of requirements specification techniques and a framework on how these techniques can be applied for system requirements engineering in a service-oriented environment. The survey is thereby focussed on the relationship between use case based specification techniques, services, and business processes that aim how to understand how services can be used to determine processes and use cases (i.e., service-based application specification). The survey also describes specification methods and notations used for functional and non-functional (QoS) service requirements. Another important aspect addressed in this report will be the introduction of ontology-based information modelling. Ontologies are formal models of a specific domain that support the communication between human and computer based actors. This facilitates exchange and sharing of knowledge within an organization. Especially in a distributed service-oriented environment as the ASG platform, various partners or organizations would like to provide and share data or other resources to customers, for example service specifications or code components. This survey will be used as a basis to develop a requirements engineering method for the ASG development method.

1.1 Purpose of this document

The goal of Adaptive Services Grid (ASG) is to develop a proof-of-concept prototype of an open development platform for adaptive services discovery, creation, composition, and enactment. To achieve its goal, ASG addresses scientific and technological issues especially related to the field of software engineering.

Requirements engineering is one of the key areas in successful software engineering and software development. Requirements engineering deals with system requirements that are specifications of the services a system should provide, the constraints on the system and additional background information, which is required to develop the system. Requirements engineering is the systematic process concerned with the elicitation, understanding, analysis and documentation of the system requirements. Requirements engineering itself is called an “engineering” process as the process itself is used in practical and systematic way where trade-offs have to be made to find the best solution [1].

One kind of requirement trade-off that must be addressed within the ASG development methodology is the relationship between service customers and service providers. Based on semantic specifications of requested services by service customers and the semantic specifications of the registered services by service

providers, ASG has to discover appropriate services, composes complex processes and – if required – generates software to create new application services on demand. Subsequently, application services will be provided through the underlying computational grid infrastructure based on adaptive process enactment technology.

In the remainder of this section, we will introduce the ASG specific terminology for services and provide the furthermore used definition for this document. In addition, we will shortly illustrate the vision for the ASG platform and stress the need for requirements engineering techniques in the ASG development methodology.

1.2 ASG specific service terminology

Before we will introduce the requirements engineering related concepts and the various requirements engineering specification techniques in the following chapters of the document, we will explain the ASG specific terminology of service and illustrate the need for requirements engineering in the overall ASG vision.

The term service in the context of ASG subsumes specific types of services, including Web Services, Grid Services as well as legacy services, i.e., pieces of software that are already existing and that are provided with interfaces so that they can be used in the context of the ASG platform. Typically, differentiation between these types of services is not required in the context of ASG. Services in ASG have a semantic specification including functional and non-functional parameters [2].

The furthermore used definition of the term service in the rest of the document is as follows: *"A service is some functionality that can be invoked using a standard interface. It can be anything from simple requests to complex business processes."* The interface of a service is specified by a standard description language that supports the description of service functionality and the service call mechanism. Especially for service grids the language should provide the description of certain properties to support service comparison and negotiation. Due to the specification of a standard interface, services could be easily discovered and concurrently reused or rather integrated in other services/applications.

Services determine the capabilities of the platform by describing possible usage scenarios of the platform. Therefore, different stakeholders should collect possible grid services for the ASG platform development process. Typical examples for services, especially web-services are "Rent a car", "Reserve a hotel room", "phone book".

A requirements engineering development process for grid services should not only address single service development but also the development of service types. Although this issue will not be addressed by the ASG project itself, we would like to mention this aspect here, as it will have some impact on the later developed requirements engineering method. In addition to a concrete service that is provided, discovered, and executed by a customer request, we can also identify service types within the ASG development process. Service types encompass a number of similar services abstracting from their differences. The idea is to exploit the commonality and to deal with all services of a type in one way. The abstraction from service differences can be on different levels (semantic, locality, functional). This idea of exploiting commonalities and proactive usage of reusable components is the main development issue of Product Line development. Product Line Engineering [3, 4, 5] is an approach that aims at exploiting reuse potential between products developed in an organization by identifying the commonalities between the products and systematizing the variabilities.

In the following, some example service types are listed. The types consist of the elements' request (i.e., the initiation of a service), response (i.e., the return of the service result), as well as interaction (i.e., the information exchange between client and service in addition to the request and response information):

- Respond: Services which response without any request, e.g. receive an email.
- Request: Services which perform the request without any response e.g. send an email.
- Request – Response: Services which perform the request and response the result, e.g. lookup a phone book entry.
- Request - Interaction – Response: Services which perform the request by interacting with the client and response the final result at the end, e.g. the current "amazon.de" service.

These service types are used in the platform development process to simplify the reasoning about the services the ASG platform provides.

We will not go further into detail of Product Line Engineering and Requirements Engineering for Product Lines in the remainder of this document, but it will be an area of further research within the ongoing work for the development of the ASG requirements engineering methodology.

1.3 ASG vision and Requirements engineering for a distributed service-oriented environment

We will now illustrate the project vision of the ASG platform and describe the relation of requirements engineering to the ASG context. Requirements seem to be prominent especially on three different levels of the ASG platform:

- *Service customer level*
  The service customer level represents the level of the ASG platform at which service requests are raised by service customers. The users of the Adaptive Services Grid thus request a desired functionality based on his needs for fulfilling a customer related business process.
- *Service grid level*
  The service grid level represents the level of the ASG platform at which the service discovery and composition takes place based on the raised service request by a customer. On this level, the ASG platform determines which existing services can fulfill the customer requests or if new services have to be generated.
- *Service provider level*
  The service provider level represents the level of the ASG platform at which service providers semantically specify their provided services and register them to the service registry of the ASG platform so that a service can be discovered and executed to fulfill a customer request. The service specification must also include quality of service specifications to express constraints on the execution of services and to identify services that are capable of fulfilling a customer request within resource boundaries.

The following picture illustrates the three different levels of requirements of the ASG platform mentioned above.

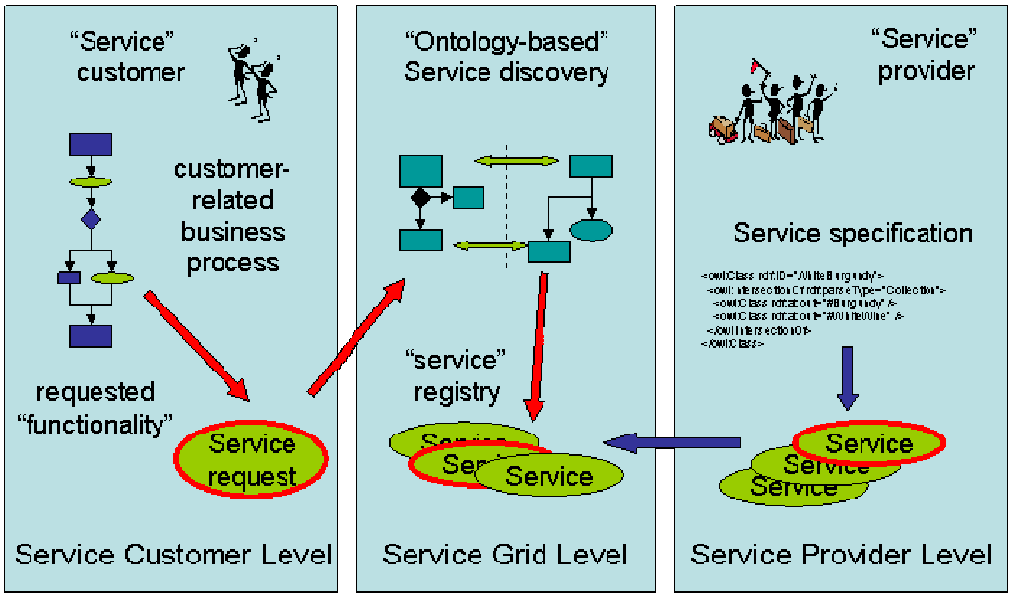


Figure 1 ASG Requirements Levels

As this survey is focused on different requirements specification techniques, the picture above illustrates the need for specific requirements specification techniques at the individual service levels. The service customer level for example requires an explicit modeling of the ongoing customer business processes and customer constraints that affect the type and outcome of the requested services. At the service grid level the need for an information model describing semantic information about the registered services is prominent. This semantic model can then be used to identify appropriate services for the specified service request. Finally, the service providers are required to provide service specifications of their services, describing functional and non-functional aspects as well as semantic usage knowledge. In the following section, we will explain the impact of specific requirements engineering techniques on these three levels and their possible usage within a holistic requirements engineering approach in more detail.

1.3.1 Service customer level requirements

The overall vision of the ASG platform is the integrated support of a service provisioning grid that supports service customers in discovering and executing of services provided by third parties. The users of the Adaptive Services Grid thus request a desired functionality. This functionality or service has to be specified on a semantic level, for example by means of semantic web technology. Semantic web technology provides concepts, languages and tools to specify the semantics of Web services based on ontologies. During this early phase of the ASG platform usage scenario, one of the most prominent needs for require-

ments engineering is raised by the user request of a desired functionality. Requirements engineering is strongly related to the elicitation of the needs of a system stakeholders. Therefore, requirements engineering techniques seems to provide suitable techniques for describing whether a service might be capable of fulfilling a specific user request or if additional modifications of services are needed.

The first requirements engineering related issue that has to be addressed during the specification of a user request is the customer related business process or goal of the service customer. The modeling of business processes primarily focuses on representing the activities that are conducted to achieve a certain business goal. These activities, which are required to achieve the business goal, are often only implicitly known by the user or customer. An explicit description of the required functionality or IT-support is not available. The goal of the business process analysis is to identify these required activities and to identify the IT-based support during the process enactment.

The requirements engineering related support of the service customer level is thus the detailed analysis and modeling of the customer's business process in order to identify the functionalities needed. With the help of a holistic mapping approach, starting from the analyzed business process and the identified services needed, a semantically specification of the service request and discovery of appropriate services seems to be possible. A holistic mapping approach requires the integrated usage of a unique representation form for requirements on all levels or at least the capability to map the requirements in a homogenous manner to a target representation form.

In addition to the customers' business processes and their goals, quality of service constraints raised by the customer must also be kept in consideration at this level. A typical quality of service requested by a customer for example is the response time of the service. Usually, customers are not willing to wait for a long period of time for the outcome of the requested service. The service planner of the ASG platform has to ensure these non-functional restrictions when composing services.

1.3.2 Service grid level requirements

After specifying the user service request and the identification of the intended service results, the ASG receives the semantically specified request and determines which existing services can fulfill these requests. Either there is a service in the ASG registry that can be used to fulfill the request. In this case, the service provision agreement is negotiated with the service provider and the service is executed by ASG, after the user accepts the contract (including non-functional parameters like response time and cost). If there is no service available to fulfill that request, the following options exist:

(1) A similar service is offered and a negotiation with the user starts;
(2) A new service is composed from existing services;
(3) A new service is generated by means of software generation methods, or
(4) the ASG platform decides that the request cannot be granted.

The two main prominent requirements related issues of this requirements level are the identification of a service capable of fulfilling the functional requirements of the user request and on the other hand, the capability to fulfill the functionality within the non-functional constraints. Ontologies seem to be a practical and suitable mean for the identification of appropriate services provided by various service providers.

Ontologies are formal models of a specific domain that support the communication between human and computer based actors. In the case of the ASG platform, that would correspond to the service customers and providers and to the ASG service registry. This facilitates exchange and sharing of knowledge within the ASG usage community. But it requires a negotiation and an agreement between the group of customers and providers on a socio-cultural level regarding the terminology and relationships used to describe the services. To be of any practical use, a well-defined notation is required for ontologies. A general accepted and well-defined language for ontologies is especially important for integrating various data sources and resources and therefore services provided by various partners.

Functional service requirements usually describe what the service will do, but how the service will do it, for example, service performance requirements, service external interface requirements, service design constraints, and service quality attributes must be modeled additionally. In order to identify a suitable set of services that can be composed many decisions must be made with respect to the environment of the requested services. On the one hand, the customer environment and his business processes are of importance and on the other hand the provider execution environments and the service development process must be considered. It is very important for the requirements engineering activities that these expectations and interests are elicited thoroughly. Therefore, non-functional requirements have to be also addressed at this level.

1.3.3 Service provider level requirements

The functional requirements of services are one of the most important types of requirements that must be specified on the service provider level. Functional requirements describe the functions that the service is to execute; for example, formatting some text or modulating a signal. A service requirements specification establishes the basis for agreement between customers and contractors or suppliers on what the product "service" is expected to do, as well as what it is not expected to do. Functional requirements are often written in natural lan-

guage, but, in requirements specification, this must be supplemented by formal or semi-formal descriptions in order to ensure the discovery of appropriate services. The selection of appropriate notations permits particular requirements and aspects of the software architecture to be described more precisely and concisely than natural language. As a general rule, notations should be used which allow the requirements to be described as precisely as possible. This is particularly crucial for safety-critical and certain other types of dependable software. However, the choice of notation is often constrained by the training, skills and preferences of the document's authors and readers.

Another important type of requirements of the service provider level are non-functional requirements or quality of services. Non-functional requirements as they are prominent in software system development also apply to services. During development, there are many decisions to be made with respect to the environment of the services, the services itself, and the service development process. These decisions not only depend on the customers' expectations, but also on the interests of other stakeholders such as developers or procurers. Thus, it is very important for the requirements engineering activities that these expectations and interests are elicited thoroughly.

An explicit usage of a common representation form for functional and for non-functional requirements throughout the requirements engineering process would enable us to create an integrated mapping of the customer related business process and thus specific "service need" to the provided services registered within the ASG registry.

1.4 Structure of this document

This survey on requirements specification techniques and the possible usage of these techniques for a service-oriented requirements engineering is structured as follows. Based on the above introduced several types of requirements and requirements levels we identified different kinds of techniques which will be explained in detail in the rest of this document. The relationship of the different requirements specification techniques and requirements levels is illustrated in the following picture:

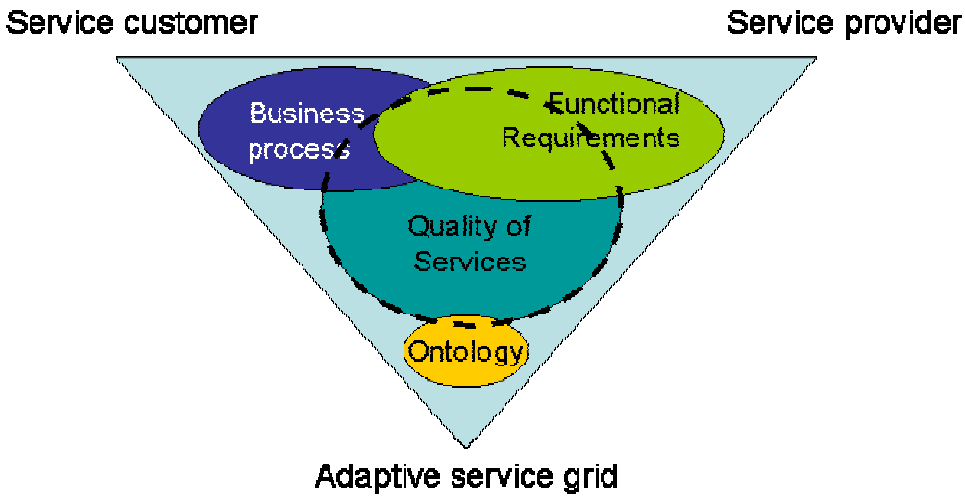


Figure 2 Requirements specification relationship

As the above picture shows, qualities of services are a central type of requirements that have to be addressed at all three levels of requirements. In chapter 2, we therefore introduce several approaches for elicitation and documentation of non-functional requirements. One specific approach [6] has been initially developed for embedded systems but shows great adaptation potential for the ASG platform and service-oriented application development.

The service providers are required to provide service specifications of their services, describing functional and non-functional aspects as well as semantic usage knowledge. Chapter 3 introduces traditional functional requirement specification techniques that can be used to express the requirements on the service provider and thus development level.

The service customer level requires an explicit modelling of the ongoing customer business processes and customer constraints that affect the type and outcome of the requested services. Therefore, business process modelling techniques will be introduced in chapter 4 of this document. The customer-related business processes and their analysis is required to identify the customer need of specific functionality and to semantically specify a service request so that appropriate services can be identified and executed.

Finally, the need for an information model describing semantic information about the registered services is prominent at the service grid level. This semantic model can then be used to identify appropriate services for the specified service request. In chapter 5 we will describe the theoretical background of semantic integration i.e. semantic models and ontologies, which are used for integrating various data sources and resources and therefore enable the discovery of services provided by various partners.

Finally, in chapter 6 we will conclude and give an outlook to the further development of a service-oriented requirements engineering process for distributed environments based on the formerly introduced requirements levels.

2 Elicitation and Specification Technique for Quality of Services

This chapter describes how non-functional requirements (NFR) or quality of service (QoS) can be elicited and documented in the context of service-oriented applications. In the following, we will use the term "non-functional requirement" to describe "quality of service" as non-functional requirements do not only apply to quality of services but also to the overall development process and thus the requirements engineering method for ASG.

2.1 Motivation

Non-functional requirements as they are prominent in software system development also apply to services. In the context of grid services or in general, of a service oriented environment, they are often called quality of services. During development, there are many decisions to be made with respect to the environment of the software, the software itself, and the software development process. These decisions not only depend on the users' expectations, but also on the interests of other stakeholders such as developers or procurers. Thus, it is very important for the requirements engineering activities that these expectations and interests are elicited thoroughly. This aspect fully applies also to the ASG vision. Many decisions must be made with respect to the environment of the requested services, on the one hand of the customer environment and his business processes and on the other hand of the provider execution environments and the service development process. Therefore, the decisions have to deal with interests of three major stakeholder groups i.e. the service customers, the platform providers, and the individual service providers.

2.2 Introduction to NFRs

In this chapter, we discuss issues in the elicitation and documentation of so called Non-functional Requirements (NFR), which essentially cover all constraints on how a system should achieve its functionality [7, 8].

The ISO Standard 9126 (2001) proposes the following taxonomy for NFRs, which can be fully applied for quality of service attributes:

- *Efficiency:* The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.
- *Portability:* The capability of the software product to be transferred from one environment to another.

- *Maintainability:* The capability of the software product to be modified. Modifications may include corrections, improvements or adaptations of the software products to changes in environment and in functional specifications.
- *Functionality:* The capability of a software product to provide functions that meet stated and implied needs when the software is used under specified conditions.
- *Usability:* The capability of the software product to be understood, learned, used, and to be attractive to the user, when used under specified conditions.
- *Reliability:* The capability of the software product to maintain a specified level of performance when used under specified conditions.

In literature and research communities several different definitions for NFR can be found. In [9] a NFR is defined as: "...in software system engineering, a software requirement that describes not what the software will do, but how the software will do it, for example, software performance requirements, software external interface requirements, software design constraints, and software quality attributes. NFR are difficult to test; therefore, they are usually evaluated subjectively". This definition is quite fuzzy. It mainly gives examples of types of NFR, but fails to classify of them. Loucopoulos & Karakostas [10] present one possible classification of NFR. They distinguish between process, product, and external requirements. Product requirements specify the desired characteristics in terms of quality attributes such as performance and security. Process requirements are constraints on the development process. External requirements are requirements that arise from external sources either within the company or outside. While working with various kinds of NFR, we experienced that this classification is not sufficient, in particular for products requirements. The ISO Standard 9126 (2001) gives a detailed classification on product requirements (see above). However, it does not give specific guidelines on how to specify or elicit the different NFR.

2.2.1 Challenges for Quality of Services

Besides the customer of a service, there are more stakeholders like suppliers, developers, and marketing. Requirements elicited from different sources are often in conflicts with each other. For example domain experts for various types of NFR frequently identify problems in the realization of requirements phrased by a naïve user. [11] particularly support the customer with their method to relate business goals to architecture.

The intertwined nature of NFR is what makes them special. Considering the impact of NFR on FR and Architectural decisions (AD) as early as possible without forcing early design decisions, and finding the right point in time and a suitable

way to treat all three of them together, is an important issue. [12, 13, 14, 15] discuss various approaches to coping with architectural issues in detail. Approaches exist that consider the dependencies between NFR and FR [16, 17, 18, 19, 20] and between FR and architecture [21, 22, 23, 24] respectively. [25] describes an approach that combines NFR and use cases [26]. Use cases and NFR are elicited separately and are then combined to make sure that the use cases satisfy the NFR.

Methods such as the Software Architecture Analysis Method (SAAM) [15, 21] and the Architecture Trade-off Analysis Method (ATAM) [27] combine NFR with AD. Both are scenario-based methods for architecture analysis. The goal of SAAM is to identify whether a software architecture satisfies its modifiability requirements expressed through scenarios. The outcome of ATAM is the risk that results from conflicting requirements and from requirements that have not been addressed in the architecture. Experiences with SAAM in case studies are presented in [28, 29]. Both methods do not help to elicit measurable NFR in an early phase, but are based on a set of scenarios. However, in practice, the elicitation of NFR, FR and AD has to be intertwined [30, 31, 32].

NFR are not only related to AD and FR, they also have internal dependencies (e.g., performance and maintainability) that have to be detected and handled. It is a challenging issue to find all these dependencies as well as solutions to overcome conflicts.

NFR are very project-specific and hardly reusable as such. Thus, specific measures are needed to support experience transfer between different projects. This becomes even harder due to the fact that NFR are often expressed vaguely and not in a quantitative way. This often gives rise to misunderstandings. For the importance of experience management, consult [33, 34, 35] provide a framework for organizing the existing knowledge about quality attributes and about the effects of architectural design decisions on the quality attributes of a software system.

Similar to the approaches above and to many other approaches [22, 11], the following approach uses goal graphs for dependencies and refinement. However, goal graphs are only used for capturing the relationships between categories of NFR such as efficiency, maintainability, but not for the actual NFR. The actual NFR are captured as part of requirements documents intertwined with FR and AD. This avoids the need to develop complicated dependencies in each project anew. Furthermore, it supports coherent documentation of NFR, FR and AD.

In general, one can say that current approaches for dealing with NFR provide a framework for thinking about NFR, but they are no help for many practical problems emerging during software development. In the following, we elaborate on these practical issues and show how to alleviate these problems

through general principles such as checklists and templates or iterative development.

2.3 Specification Techniques for Quality of Services

In this section, we introduce several approaches for elicitation and documentation of non-functional requirements.

2.3.1 The EMPRESS Method

In this section we present the EMPRESS Method [6]. It is a comprehensive method for eliciting, analysing, documenting and managing NFRs. The main features are:

- a process for common treatment of the various high level QAs (ISO 9126) e.g., maintainability, efficiency, portability, usability, security, and reliability
- experience-based quality models that capture experience with general characteristics of quality attributes (QAs), metrics to measure these QAs, and means to achieve them
- detailed elicitation guidance in terms of checklists and a prioritization questionnaire. The former are derived from the quality model and the types of QAs and help to elicit efficiency NFR in concert with use cases and a high-level architecture. The latter helps to focus the effort spent on NFR elicitation on the most important Qas.
- documentation guidance by providing document structure and templates
- treatment of NFR together with functional requirements and the system architecture
- requirements management support including a dependencies-analysis on the elicited set of requirements.

The EMPRESS method uses a notation based on goal graphs for dependencies and refinement. We use this notation for capturing the relationships between categories of NFR such as efficiency and maintainability in reference quality models. The actual NFR are captured as part of requirements documents intertwined with functional requirements (FR) and architectural decisions. This avoids having to develop complicated dependencies in each project from scratch by capturing the experience with dependencies in reference quality models. Furthermore, it supports coherent documentation of NFR, FR and architecture.

The method distinguishes between quality attributes (QAs) which are captured in quality models (using goal graphs), and non-functional requirements (NFR)

that are captured in documents based on templates. A QA is a non-functional characteristic of a system, user task, system task, or organization. QAs of the organization include development process specific aspects. A NFR describes a certain value (or value domain) of a QA that should be achieved in a specific project. The NFR constrains a QA by determining a value for a metric associated with the QA. For example, the NFR "The database of our new system shall handle 1000 queries per second" constrains the QA "workload of database". The value is determined based on an associated metric "Number of jobs per time unit".

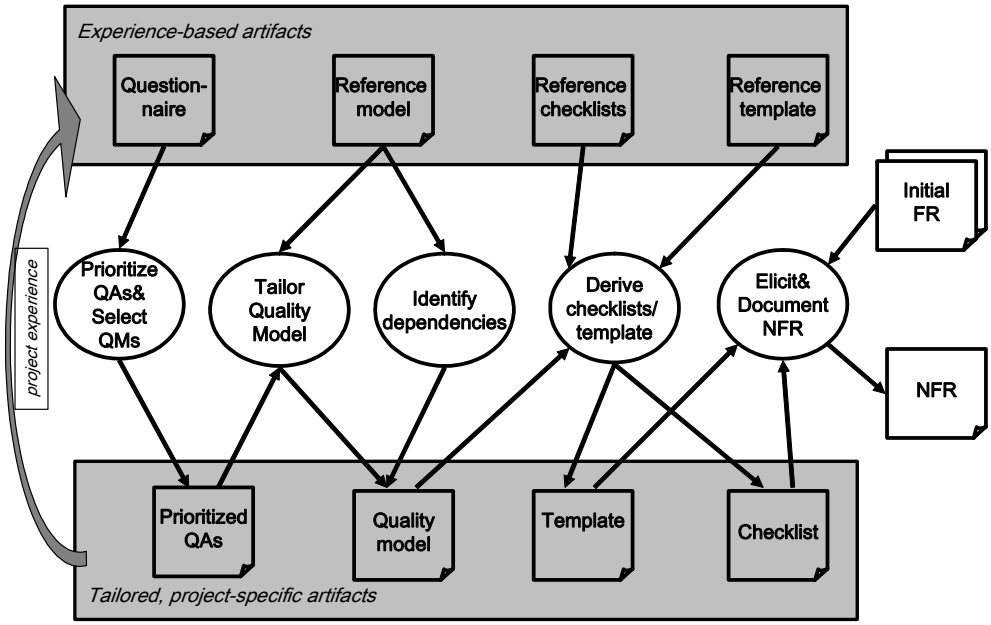


Figure 3 The EMPRESS Method process and artefacts

Figure 3 shows the main artefacts used and produced in our method. In a first step, the prioritisation questionnaire is used to focus the elicitation and documentation process. This is performed to prioritize the high-level QAs (i.e., maintainability, efficiency, reliability, usability). For the highest ranked QAs, the reference quality models get tailored in a workshop. In this tailoring process, experts from the company tailor each quality model to the needs of the project. Based on the tailored quality models, the reference checklists and templates are tailored to specific project contexts. The tailored checklists and templates are then used in a second workshop for the actual NFR elicitation.

The experience-based artefacts were initially developed from literature. In each project, the experience during the tailoring was incorporated into the reference quality models. Therefore, each application in a project resulted and will result in more mature reference quality models.

2.3.2 The soft-goal notation

Another comprehensive approach for dealing with non-functional requirements is presented by [9].This NFR framework, provides detailed guidance on how to refine NFR and how to denote relationships between the NFR. The focus is on refinement and relationships between different NFRs. The major step steps when applying the soft-goal notation are:

- Acquiring knowledge about domain system, as well the functional aspects
- Identifying the important NFRs
- Refinement and decomposition of the NFRs
- Identifying and selecting operationalization

In Figure 4, an example is given of the soft-goal notation. With the help of soft-goals, the non-functional aspects of the system can be modelled and analysed.

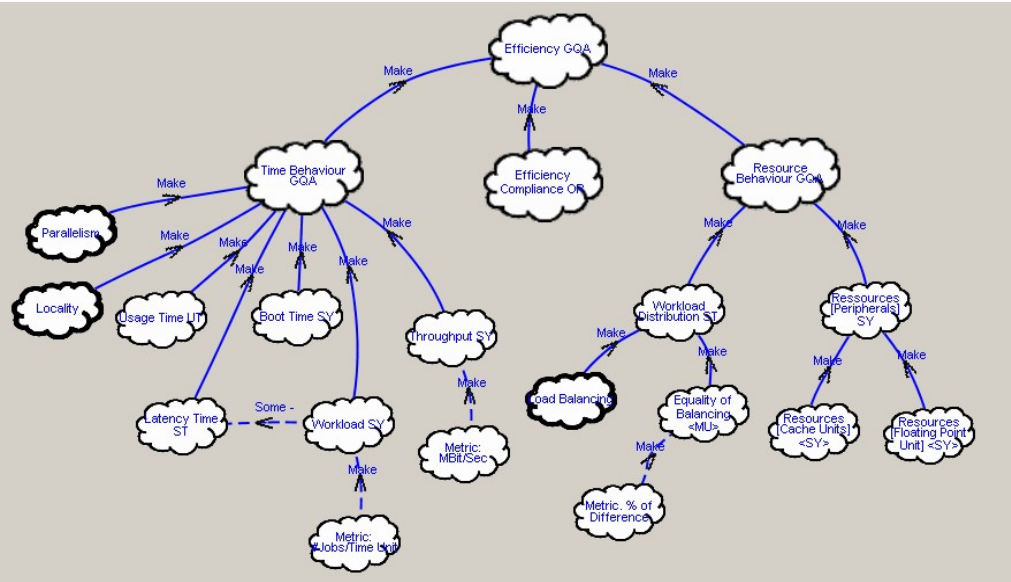


Figure 4 Example of the soft-goal notation

The soft-goal notation is more formal, in the sense that the there is a semantic in the arrows and how the decomposition of soft-goals are performed. It is possible to express several types of relationships, e.g. decomposition, and, or, positive or negative influence, etc.

As mentioned, the focus is on how to refine and operationalize the requirements. However, the method offers little support for how to elicit and document the requirements. The soft-goals, therefore, needs to be complemented with a process and documentation support around the notation.

2.3.3 Specialized methods

Besides the two general methods mentioned in Section 2.3.1 and Section 2.3.2, there exist many specialized methods, dealing with one specific quality aspects, e.g. security or usability. Sometimes if there is a particular focus on one particular quality of the system, the general methods are not enough. Even though they provide a comprehensive and integrated approach for working with the requirements, having a specialized method will be more appropriate for the important aspect. However, the challenge then is to integrate, for example, safety issues into the normal requirements process. This is not done without effort. As this report is focused on requirements methods, the specialized methods are not further elaborated on here.

2.4 Quality of Services and ASG

The approach described in section 2.3.1 has been initially developed for embedded systems but shows great adaptation potential for the ASG platform and service-oriented application development. The approach introduced here [6] is based on the explicit usage of use cases for describing requirements and on quality models derived from the ISO 9126 standard, as well as general problems and challenges when working with NFR. Requirements in general and NFR in particular are subjective, have many stakeholders and are often conflicting. The approach presented includes processes for prioritizing quality attributes that are important to a specific context, eliciting NFR and identification and analysis of dependencies among the NFR. The aim is to provide an experience-based approach that facilitates efficient and effective elicitation and documentation of NFR. Having a structured method that aims at providing measurable, traceable and focused requirements, rather than having ad-hoc and ambiguous ones achieves this. The approach uses use cases as main technique, though the general principle of having a structured and experience-based process is applicable to other techniques as well. The usage of use cases makes it easier to develop an integrated requirements engineering process, as use cases are a prominent specification technique for functional behaviour of software products and can also be used for describing business processes of service customers.

3 Specifications for functional Requirements

Functional requirements can be specified using a wide range of notations, with different levels of formality. The selection of notation depends on many factors, such as target audience, domain and type of process. For example, it the target audience do not have a technical background, choosing a natural language based notation is probably better than using UML or Petri nets. On the other hand, if you are developing safety-critical systems, it might be necessary to ensure the quality at an early stage to be able to live up to the required quality standard.

3.1 Motivation

The functional requirements of services are one of the most important types of requirements that must be specified on the service provider level. Functional requirements describe the functions that the service is to execute; for example, formatting some text or modulating a signal. A service requirements specification establishes the basis for agreement between customers and contractors or suppliers on what the product "service" is expected to do, as well as what it is not expected to do. Functional requirements are often written in natural language, but, in requirements specification, this must be supplemented by formal or semi-formal descriptions in order to ensure the discovery of appropriate services. The selection of appropriate notations permits particular requirements and aspects of the software architecture to be described more precisely and concisely than natural language. As a general rule, notations should be used which allow the requirements to be described as precisely as possible. This is particularly crucial for safety-critical and certain other types of dependable software. However, the choice of notation is often constrained by the training, skills and preferences of the document's authors and readers.

3.2 Specification Techniques for Functional Requirements

This section presents an overview of some requirements specification notations. In Section 3.2.1, use cases and scenarios are discussed. State-based notations are discussed in Section 3.2.2. Section 3.2.3 elaborates on sequence-based notations. Finally, in Section 3.2.4, an overview of UML is presented.

3.2.1 Use cases and scenarios

A use case is a description of how end-users will use the system. It describes a task or a series of tasks that users will accomplish using the software, and includes the responses of the software to user actions. Use cases may be included in the Software Requirements Document (SRD) as a way of specifying the end-users' expected use of the software.

Examples for use cases of the ASG platform are:

- Managing the workload of the grid
- (Un-) mounting devices
- Resource monitoring
- Service (un-)registration

In addition to these use case for the ASG platform, the functional behavior of service specific use cases must be modeled.

- Service-specific "use case"-oriented functional specifications

The purpose of use cases is to elicit all possible uses of the ASG platform.

A scenario is a set of one or more typical interaction dialogs between the users of a system (people or other systems) and a proposed system that is about to be developed. Scenarios are developed to assist in understanding business events, objects and interactions. Scenarios document specific transaction sequences, transformations, interfaces and information exchange. Use case scenarios facilitate communication between the people who request a system, analysts, developers and testers. They are used to validate understanding, and to identify normal and special use situations. Scenarios clarify an evolving agreement between requesters and development teams.

For the example use case service registration, the scenario would be as follows: The grid administrator manages the registration process and announces the new service to the service pool, accessible by all grid users.

Since scenarios describe activities that the future system is involved in, they also postulate architectural requirements and constraints. In regard to the ASG platform, they support the development process for the architecture by revealing requirements for all kinds of interaction with the platform.

Unfortunately, due to the wide range of applications and usage situations, the terminology is anything but standardized. The following definition is used here:

- A use case is an abstract task performed by an actor. For example "Set seat" by "internet user". This is typically visualized using UML use case diagrams, see Section 0.
- A scenario is a detailed description of the interaction between the system and the actor(s), describing one possibility of many. It is important to remember that the scenarios are not complete in the sense that they cover all variations or requirements, see table below. Also, in the flow of events, the sequence is usually not mandatory, in the sense that things have to happen exactly in that order.

Table 1 Example of scenario

Scenario	Set saved seat position according to internal identification
Actors	Driver
Intent	Actor sets saved seat position
Pre conditions	None
Flow of events	1. Actor enters identification over internal identification input 2. System moves driver seat in the seat position saved under the entered identification [Exception: no seat position saved] [Exception: speed is too high] [Exception: actor activates identification input] [Exception: actor activates direct seat position input] [Exception: technical problem]
Exceptions	• No seat position saved: system doesn't react • Speed is too high: system doesn't react • Actor activates identification input: at internal identification input: → UC *„Set seat position according to internal identification input",* at external identification input: → UC *„set seat position according to external identification input"* • Actor activates direct seat positioning: → UC *„Set seat position according to direct seat position input"* Technical problem: system sets at least one movement direction not as wished
Rules	• Speed too high, if > 5 km/h • At set saved seat position: first the relaxing movements, then the opposed movements. Relaxing movements are increasing the distance between seat and steering-wheel, increasing the back angle, lowering the seat area, increasing of the casing size • At a particular time, the seat is only allowed to move in two directions. The order is: distance between seat and steering-wheel, back angle, casing size, seat area front, seat area back
Quality constraints	None
Monitored environmental variables	• Internal identification input • Actual speed • Saved seat position
Controlled environmental variables	• New seat position
Post conditions	Seat position changed

There exist several different templates and standards for how to write the scenarios. One of the most used is the one from Cockburn [26]. The example in Table 1 is an extension of the Cockburn templates, in an attempt to adapt the scenario description to the automotive industry [37]. Lauesen also presents several variations of scenarios in [38].

There is also a concept of misuse cases [39]. A misuse case is a use case with an actor that wants to misuse the system one way or another. Misuse cases are particularly good when eliciting security and safety requirements, as the threat or risk are explicitly dealt with.

The main benefits of natural-language based notations are that they are expressive and compact, making them appropriate for communication with non-technical stakeholders. The drawbacks are that they can be ambiguous and is difficult to perform any kind of automatic analysis.

3.2.2 State-based notations

There exists a range of state-based notations, e.g. Harel's state chart [40], SCR [41] and UML state charts [42]. The basic building blocks in the state notations are states and transitions between the states, see Figure 13.

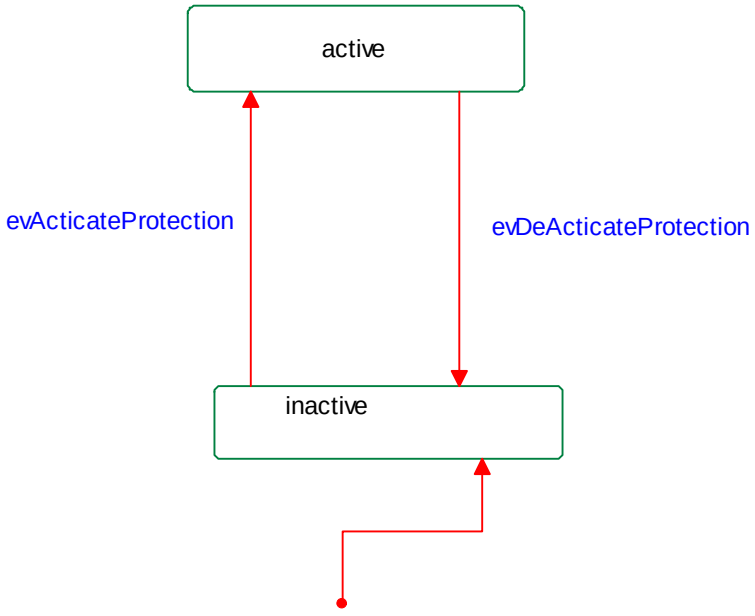


Figure 5 Example of State chart

A state-based notation is of a more formal nature than use cases. That means that it is possible to perform automatic checks directly on the state charts, as well as generating design, code or even test specification. On the other hand, a state model is often not understandable to a non-technical user and is therefore not as appropriate when communicating with customers.

SCR is essentially a state-based notation, though it looks quite different from other state-based notations and has a somewhat different semantic [41]. The core in SCR is transition table, see Table 2, where mode means the same as state. A special feature in SCR is that the mode transition tables (and event tables) can be arranged hierarchically, making SCR more scalable.

Table 2 SCR example

Old Mode	Event	New Mode
TooLow	@T(mPres ≥ Low)	Permitted
Permitted	@T(mPres ≥ Permit)	High
Permitted	@T(mPres < Low)	TooLow
High	@T(mPres < Permit)	Permitted

The UML state charts [42] is basically an extension of the Harel state charts [40].

3.2.3 Sequenced-based notations

Sequence-based notations can be seen as a formalization of scenarios, see Section 3.1. As with scenarios, the focus is on the interaction between the system and the actors. Examples of sequence-based notations are Message Sequence Charts [43], Sequence Charts in UML [42] and sequence-based software specification [44].

The benefits of sequence-based notations over scenarios are that they are less ambiguous and have some possibility for automatic analysis. The drawback is that the expressiveness is limited to the events and their sequence.

A sequence consists of one or more events (arrows in Figure 14) being sent between one or more actors or system components (vertical lines in Figure14). Sequence-based notations can be used at different levels of abstraction. They can be useful to as a formalization of scenarios, to get to a less ambiguous model. They can also be used, as in UML, to specify communication sequence between components. Normally, sequence-based notations are used to specify sequences between system components, i.e. a relative detailed level of abstraction.

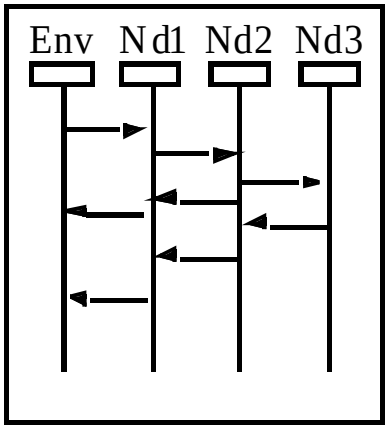


Figure 6 MSC example

In the formalized version of sequence-based notations by Prowell and Poore, a lot of automatic analysis is possible sequence-based software specification [44]. They do however lack a graphical notation. In principle, the ideas from Prowell and Poore are similar to those of SCR, see Section 3.2.2.3.

3.2.4 Unified Modelling Language (UML)

The Unified Modelling language (UML) is basically a collection of several notations and models [42]. It originated from the object-oriented community, but has developed beyond this and is a generally applicable framework. UML contains a number of different diagrams:

- Use case
- Class
- Object
- Component
- Deployment
- Sequence
- Collaboration
- State charts
- Activity

Several of these have already been mentioned in previous sections. The main type of diagram not mentioned is those used for the static structure, namely class, object and component diagrams. However, from a requirements point of view, they are usually not of any interest.

Even though UML is usually used for design, there are benefits of using UML to specify requirements as well. Having a unified modelling framework makes it easier to integrate different issues in the development process. Also, there are a lot of tools for UML. The drawback is that UML is mainly a design framework. Hence, it can be used for detailed requirements specification, but is not appropriate for customer-specific issues. Also, for a non-technical reader, UML is more difficult to understand than scenarios.

Especially the existence and usage of use cases within the UML enables an integrated and holistic usage of use cases in the requirements engineering approach for business processes, functional and non-functional requirements.

As the UML is currently widely known and used in industrial practice we will not further introduce the UML notation here but refer to literature [42].

3.3 Functional Specifications for ASG

The usage of an integrated and holistic usage of a uniform and consistent representation form for the various requirements on the three levels seems to be necessary to develop a requirements engineering process for the ASG platform. Use cases are a prominent specification technique for functional behaviour of software products and can also be used for describing business processes of service customers.

Within the ASG development, the following existing template for use cases is used by the various partners to specify their use cases:

3.3.1 Actor

Enter the main actor of the interaction that has an interest in performing the functionality described in the interaction. An actor can be either human (e.g., an external human service consumer) or not (e.g., an external service provider). Also, an actor can be external to the ASG platform or internal (e.g., an ASG platform subsystem that interacts with another internal subsystem).

3.3.2 System

Enter the system involved in the interaction with the actor. In the case of the ASG project, this can be either an the ASG platform accessed by an external actor or a subsystem of the platform accessed by another subsystem (in this case, the accessing subsystem plays the role of the actor) or by an external actor.

3.3.3 Description

Provide a brief description of the reason for this interaction or a high-level description of the action that implicates this interaction.

3.3.4 Scenario

Explain the actor/system interaction (functionality) in form of a structured flow of events. This also includes a description of the information and the type(s) of data that are involved in this interaction.

3.3.5 Interface requirements

Specify what kind of interface is needed for this interaction. interfaces that have been identified, for example, for the Registry are e.g. query interface, interface for data management (service registration,…). The type of interface depends on the kind of communication between the actor and the system.

3.3.6 Non-functional requirements

Identify any non-functional requirement that may need to be addressed during design or implementation. These may include performance, reliability, security issues or any additional quality attributes. Please, define the requirements as (measurable) attributes referred to single steps of the flow of events described in the scenario (e.g., step 3 must not take longer than 3 ms, or data sent during step 5 must not be visible to any party other than the system).

3.3.7 Includes

List any other interactions, work components or modules that have some kind of relationship or dependence to this interaction.

3.3.8 Frequency of Use

Estimate the number of times this interaction will be performed or the point in time (where during platform execution) when this interaction takes place. For recurring interactions the time interval between communications or events that trigger this interaction may be specified.

3.3.9 Assumptions, constraints

Assumptions or constraints that were made in the analysis that led to this interaction description should be specified as preconditions, post-conditions, or minimal guarantee.

- Preconditions

Enter the system state that must be achieved before the interaction can be started.

- Postconditions

Enter the condition that holds after the interaction was performed successfully.

- Minimal Guarantee

Enter the minimal system state that must be achieved after the interaction even if the interaction does not perform successfully.

3.3.10 Extensions

Extensions define variations or exceptions that can happen during the main success scenario. Please, specify the type of the extension. Three alternatives are differentiated: Option (i.e., a further interaction that may take place), Alternative (i.e., a set of different interactions from which one must take place), Exception (i.e., a further interaction that takes place in the case of an exception).

3.3.11 Detailed Functions

Enter a list of all detailed functions that are relevant for performing the interaction functionality.

3.3.12 Notes and Issues

List any additional comments about this interaction or any remaining open issues or TBDs (To Be Determined) that must be resolved. Identify who will resolve each issue, the due date, and what the resolution ultimately is.

This explicit usage of such a kind of use case template throughout the requirements engineering process enables us to create an integrated mapping of the customer related business process and thus specific “service need” to the provided services registered within the ASG registry.

4 Customer oriented Business Process Modelling

Two modelling notations commonly used to describe requirements from a customer's point of view are business processes and Use Cases. The following section will focus on an in-depth introduction to business process notations. Regarding a detailed description of the Use Case notation please refer to Section 3.2.

Commonly business processes are used to specify requirements on a higher, abstract level while Use Cases are used to refine the business processes and detail the described requirements.

4.1 Motivation

Business processes can be used to document requirements from a customer's point of view. A business process represents a sequence of activities performed to achieve a certain goal. Modelling of business processes primarily focuses on representing these activities.

A business model increases common understanding of requirements, as the model can serve as a basis for communication for all stakeholder groups. As such a business process model is usually easily understandable it does not require any previous experience with the notation.

In requirements engineering business processes are used in several different ways. On the one hand they are sometimes seen as being only surrounding information to the requirements process that has to be kept in mind while writing the requirements. On the other hand business processes are used to actually specify requirements on an abstract level describing process activities. Business process can be a starting point w.r.t specifying functional requirements on a more abstract level, thus being an appropriate mean to describe customer oriented requirements.

4.2 Business process modelling notations

The following section introduces several notations that can be used to document, model a business processes:

- EPC
- BPEL

- Activity Diagrams
- BPML

4.2.1 EPC

EPC or Event-Driven Process Chains are used to represent business processes and are one of the key concepts of the ARIS product from IDS Scheer [45]. The primary incentive of EPCs and the representation of business processes through EPCs was to increase the efficiency of the represented business activities through introducing a systematic flow and increasing the parallel execution of these activities.

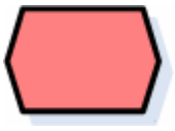
The following section introduces the different concepts of EPCs used to document business processes.

4.2.1.1 Function

A function represents an activity that has to be performed by a certain organizational unit to reach a given goal. Examples of functions are:

- Place order
- Chose article
- Chose payment method

4.2.1.2 Event



An event formally represents a condition that relates to a certain function. An event may represent a pre-condition or a post-condition to a certain event. Examples of events are:

- Login successful
- Payment method chosen and accepted

4.2.1.3 Logical operators

EPC offers three logical operations that can be used to relate events and functions:

- XOR : one and only one of the alternatives may be chosen
- OR: one or more alternatives may be chosen
- AND: all alternatives have to be executed

4.2.1.4 Process path

A process path is used to represent a process part (an activity) that is further refined/described through a second EPC diagram.

4.2.1.5 Organizational unit

An organizational unit represents a role, usually a group of persons from the company related to the business process. Examples are:

- Customer
- Financial department
- Marketing

4.2.1.6 Information unit

An information unit represents a certain object that is needed as input or produced as output during a certain activity or a set of activities. Examples are:

- Document
- Products

4.2.1.7 Example of a business process modelled with EPCs

The following broadly sketches a business process using EPCs. The process reflected is a basic order process. A customer logs into the system and chooses one or more articles that he wants to order. After selection and acceptance of a payment method the order is placed. Figure 15 shows this process.

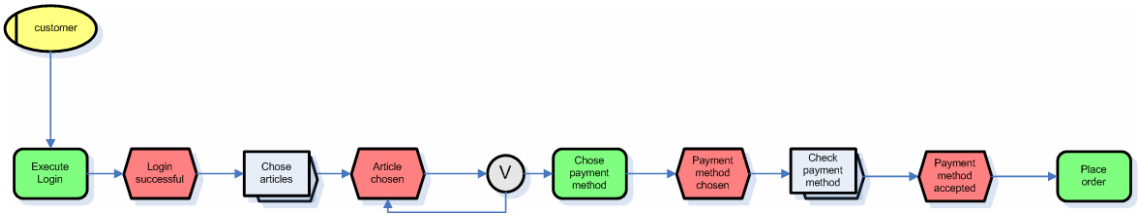


Figure 7 Order process

4.2.2 Activity Diagrams

Activity Diagrams are a subset of the Unified Modelling Language used to represent a flow of activities. Using notational elements like activities, splitting/synchronization, decisions and responsibilities Activity Diagrams can also be used to model business processes. As compared to Event-Driven Process Chains, an Activity Diagram does not contain an event element, thus representing events implicitly through begin/end of an activity. The following figure represents an example of a business process using Activity Diagrams.

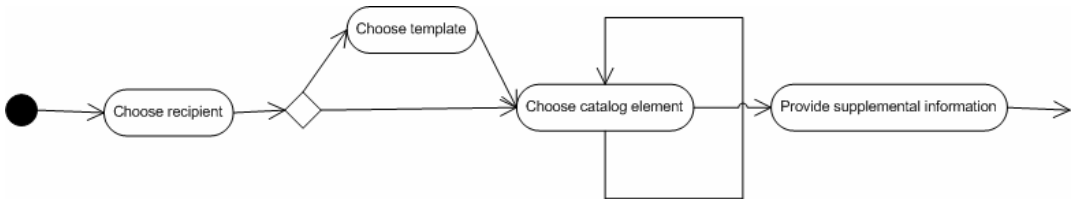


Figure 8 Excerpt of Order process using Activity Diagrams

Activity Diagrams, as a part of the UML standard, are widespread. The notation is easy to learn and understand, thus very appropriate to the context of customer oriented requirements engineering.

4.2.3 BPEL

BPEL, the Business Process Execution Language for Web Services (BPEL4WS) [46] is an XML-based notation that is used to represent business processes orchestrating web services. Historically BPEL combines IBM's Web Services Flow Language (WSFL) [47] and Microsoft's XLANG [48] specification.

BPEL is used to combine several services (usually Web Services) to form a business process as depicted in Figure 16. Based on a WSDL-specification [49] that represents the functionality offered by a web service, the service is integrated, as an activity into a more general flow of activities representing a business process.

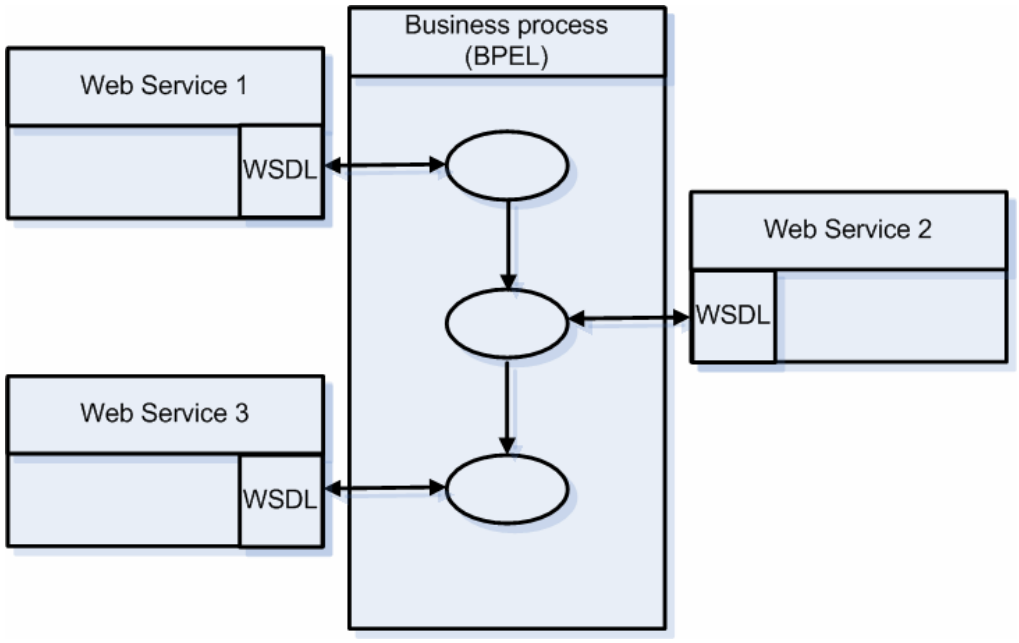


Figure 9 BPEL process

The following section introduced the structure of a BPEL document and explains the different elements forming the structure.

4.2.3.1 Process

A BPEL document consists of a general element process. This XML-tag represents the business process that is modelled and adheres to the following structure:

```
<process…>
    <partnerLinks>…</partnerLinks>
      <variables>...</variables>
    <correlationSets>…</correlationSets>
      <faultHandlers>…</faultHandlers>
      <compensationHandlers>…</compensationHandlers>
      <eventHandlers>…</eventHandlers>
    Complex or basic activities
</process>
```

4.2.3.2 partnerLinks

The interaction of the described business process with external web services is represented via the tags partnerLinks and partnerLinkTypes.

A partnerLinkType element represents the interaction between two entities on a more abstract level, defining a relationship by indicating the different roles participating to the relationship.

```
<partnerLinkType name ="XXX" …>
    <role name =""> …</role>
    <role name =""">…</role>
<partnerLinkType>
```

The partnerLink element can then be used in a business process to describe that a speciric relationship with a web service exists, indicating the role of the business process and the one of the Web Service:

```
<partnerLinks>
<partnerLink    name=""    partnerLinkType="*reference*"    myRole="role of business process"? partnerRole="role of web service partner">
</partnerLink>
</partnerLinks>
```

Example:

```
<partnerLinkType name ="ValidationCC" …>
    <role name ="OrderShop"> …</role>
    <role name ="CreditCardValidation">…</role>
<partnerLinkType>
```

```
<partnerLinks>
<partnerLink name="VCC" partnerLinkType="ValidationCC" my-
Role="OrderShop"? partnerRole="role CreditCardValidation">
</partnerLink>
</partnerLinks>
```

4.2.3.3 Scope

A scope is an abstract element enabling a context specific execution of one or more activities. The process element implicitly also forms a scope. As compared to a process a scope can provide faultHandlers, eventHandlers, compensation-Handlers as well as variable definitions. These different elements only apply to the scope, they are not visible to the overall process or other scopes.

```
<scope ...>
<variables>...</variables>
    <correlationSets>…</correlationSets>
<faultHandlers>…</faultHandlers>
<compensationHandlers>…</compensationHandlers>
<eventHandlers>…</eventHandlers>
    Complex or basic activities
</scope>
```

4.2.3.4 Variables

Variables enable a business process to store (and process) specific data values for example in order to handle internal or global process states.

```
<variable name ="" messageType="" type ="" element="">
```

4.2.3.5 FaultHandlers, EventHandlers, CompensationsHandlers

BPEL provides several handlers to react to different events providing the means to execute the necessary actions in reaction to these events.

CompensationHandler

A compensationHandler provides the necessary steps that are to be executed in order to reverse certain actions.

<compensationHandler>

Complex or basic activity

</compensationHandler>

FaultHandlers

A fault handler is equivalent to the try catch elements in Java. An activity or a scope is delineated by a faultHandler providing the ability to react to faults occurring during the execution of the respective scope or activity.

```
<faultHandlers>
<catch faultName="">
Complex or basic activity
<catch>
...
</faultHandlers>
```

EventHandlers

EventHandlers can be associated with scopes or the process itself and provide a means to react to different sorts of events as for example:

- Alarms
- MessageEvents

If an Alarm (timeout) occurs or a specific message arrives, the EventHandler element allows to react to these situations by indicating a set of complex or basic activities that are to be executed.

```
<eventHandlers>
    <onMessage ...>
    Complex or basic activity
    </onMessage>
    <onAlarm ...>
    Complex or basic activity
    </onAlarm>
</eventHandlers>
```

4.2.3.6 Activities

The most important and central part of BPEL are activities. BPEL distinguishes structured and basic activities. Structured activities determine the flow of activities and thus contain basic activities as sub elements.
Structured activities are:

- Sequence
- Switch

- While
- Pick
- Flow

Sequence

A sequence executes two activities one after each other determining the order of execution.

```
<sequence>
    <invoke …>
    <invoke…>
</sequence>
```

Switch

The switch element is used to introduce conditional flow of activities.

```
<switch>
<case condition="booleanexpression">
    Structured or basic activity
</case>
<otherwise>
      Structured or basic activity
</otherwise>
</switch>
```

While

The while activity enables a looped, repeated execution of several activities.

```
<while condition="Boolean-expression">
    Structured or basic activity
</while>
```

Flow

As opposed to sequence, flow provides the ability to execute activities concurrently providing a means to synchronize these.

```
<flow>
    Structured or basic activity
</flow>
```

Basic activities are:

- Invoke
  Invokes an operation of a partner.
- Receive
  Message from a partner is received and processed.
- Reply
  Reaction to a reply element.
- Assign
  Assigns or copies values to variables.
- Terminate
  Terminates the whole process
- Throw
  If a certain state (error) is reached the throw activity can initiate the necessary steps to be taken by signalling the state through the throw activity.
- Wait
  Execution is stopped for a certain time period.
- Empty
  Does nothing.

4.2.4 BPML

BPML, the Business Process Modelling Language is an XML-based notation, presented by the Business Process Management Initiative (BPMI) [50], used to represent business processes. BPML is similar to BPEL4WS, but, as compared to BPEL does not focus on the orchestration of web services. Using elements like process, action, sequence, while, switch, fault handler, exception handler among others, it can be used to represent business processes. Due the fact that these business processes are represented as XML documents, they are hard to understand. The notation is not easy to learn and usually requires expert knowledge. Using the Business Process Modelling Notation (BPMN) [51], a graphical representation of a BPML document, this disadvantage can be weakened considerably.

4.3 Comparison and usage of BPEL and EPC for ASG

EPCs do not require a high experience to make use of them. The notation is quite simple and straightforward. It provides an excellent basis to document customer requirements as the customer can easily understand the provided models. BPEL on the other hand requires in depth knowledge of its structures to make reasonable use of it. The training curve is especially high in the begin-

ning as BPEL is not a very easy to learn specification notation. In addition customers may experience difficulties to document their requirements within a BPEL document or to understand the requirements documented within the XML document. To weaken this disadvantage the Business Process Management Initiative (BPMI) [50] provides the Business Process Modelling Notation (BPMN) Concept [51], a graphical notation to model a business process (integrating several web services) that can be transformed into a BPEL document. This notation was released in a first version and still experiences several difficulties but can nevertheless be used to clarify a BPEL XML document.

BPEL can be used to address specific services while specifying the requirements. If the customer explicitly requires a specific service that he wants to use and integrate, BPEL provides an excellent basis to address this requirement.

BPEL is a very formal representation of a business process that does not leave much flexibility in its description thus being much more unambiguous as EPCs for example. EPC represents a semi-formal notation that can usually be interpreted in several different ways making it hard to predict the intended specification.

Based on this formal character BPEL provides another advantage that can be crucial especially in the domain of adaptive service grids. BPEL is an executable specification, the XML code can be executed by a dedicated server as for example Intalio n3 [52] not requiring much effort regarding implementation [53].

4.3.1 Usage of BPEL to specify functional requirements

BPEL can be used by service providers to provide new services that are based on existing web-services. With the help of BPEL the interaction of different services can be described forming a business process that represents a new service that may be published and invoked.

5 Specifications for Ontology-based modelling

In this chapter, we will describe the theoretical background of semantic integration i.e. semantic models and ontologies. Later on, we will introduce state-of-the-art notations and specification techniques for ontologies as the focus of this document is on specifications for service-oriented requirements. At the end of this chapter, we will discuss the impact of ontologies for a distributed service-oriented environment in more detail.

5.1 Motivation

Heterogeneity is the major challenge for integrating various data sources and resources of different partners. Especially in a distributed service-oriented environment as the ASG platform, various partners or organizations would like to provide and share data or other resources to customers, for example service specifications or code components. Many state-of-the-art information integration techniques exist today, but none of them has prevailed in industry so far. Today, information integration is mostly done on a syntactical level, but semantics for information preservation and transformation is missing.

5.1.1 Specifying relationships between models

The underlying idea of semantic models is the ability to associate a unique meaning with every information element. By doing this, information is transformed into knowledge, which is a fundamental precondition for automatic information processing. Based on this goal, semantic models should feature the following three characteristics:

5.1.1.1 Shared terminology

The meaning of information can only be explicitly distinguished if every involved partner has the same language. This is typically realized by assigning a specific and unique identifier to every element. Knowledge manipulation can then be achieved by sheer symbolical transformation. This approach can only be successful if the application domain is well-defined and every domain element can be identified. This can be achieved by developing a domain-specific terminology for each domain. To combine the individual domains matching elements between different domains must be made explicit.

5.1.1.2 Relationships between elements

In general, it is not sufficient to assign a unique identifier to an element. Knowledge is principally gained not until the relationship between elements is understood. For that reason, semantic models provide modelling primitives to express the role of the identifiers. Typically, modelling primitives are object-oriented i.e. it is possible to express that an identifier refers to a specific term. This means that an identifier can be treated as an abstract element that describes the characteristics of a concrete domain element.

5.1.1.3 Mathematically based semantic

As semantic models provide the capability to combine modelling primitives, it is important to define the semantic of these primitives in distinct and mathematically correct expression. This is realized by a well-defined interpretation of every construct.

5.2 Introduction to Ontologies

Models that hold the above mentioned three characteristics are generally called ontologies. Many modelling paradigms feature only the first two characteristics, i.e. the ability to uniquely identify elements and to specific relationships between model elements. In contrast, the third characteristic, i.e. the model-based semantic, is typically found only in semantic models.

Ontologies are formal models of a specific domain that support the communication between human and computer based actors. This facilitates exchange and sharing of knowledge within an organization. But it requires a negotiation and an agreement between a group of users on a socio-cultural level regarding the terminology and relationships. Object-oriented modelling approaches are advancing by becoming a part of a knowledge management system and therewith can be used at runtime. These aspects are formalized in the following definition of the term ontology: [54]

"*An ontology is an explicit specification of a shared conceptualization.*"

To be of any practical use, a well-defined notation is required for ontologies. A general accepted and well-defined language for ontologies is especially important for integrating various data sources and resources and therefore services provided by various partners. During the last decades many different notations for ontologies have been developed but none of them has prevailed. In the following sections of this document we will introduce the most prevalent representatives of existing notations for ontologies. After that, we will discuss the

impact of ontologies for a distributed service-oriented environment in more detail.

Ontologies are a key enabling technology for the semantic web. They interweave human understanding of symbols with their machine-processability. Ontologies were developed in Artificial Intelligence to facilitate knowledge sharing and reuse. Since the early nineties, Ontologies have become a popular research topic. They have been studied by several Artificial Intelligence research communities, including Knowledge Engineering, Natural Language Processing and Knowledge Representation. More recently, the concept of Ontology is also becoming widespread in fields, such as intelligent information integration, cooperative information systems, information retrieval, electronic commerce, and knowledge management. The reason ontologies are becoming so popular is largely due to what they promise: a shared and common understanding of a domain that can be communicated between people and application systems. In a nutshell, Ontologies are formal and consensual specifications of conceptualizations that provide a shared and common understanding of a domain, an understanding that can be communicated across people and application systems. Thus, Ontologies glue together two essential aspects that help to bring the web to its full potential:

- Ontologies define formal semantics for information, consequently allowing information processing by a computer.
- Ontologies define real-world semantics, which make it possible to link machine processable content with meaning for humans based on consensual terminologies.

5.3 Specifications Techniques for logical relationships

With ideas and interests in semantic web technologies, ontologies have realized a boom during the last years. Unfortunately, this has not contributed to the definition of the term ontology. In many cases, ontology-based structures are just controlled glossaries or dictionaries like classifications or thesauri. The ability of relating relationships (in RDF called „Reification“) and applying rules is seldom used due to its complexity. Although this is one of the main unique characteristics of ontologies compared to other system of concepts.

An ontology can be compared to a database: It consists of two complementary parts, the structure (database schema) and the content (data).

Typical languages for ontolgies are for example RDF, DAML+OIL, KIF, F-Logic, OWL or WSMO. Thereby, RDF and OWL are semantic web languages and are characterized as „W3C Recommendation“ by the World Wide Web Consortium (W3C). Thus, they are classified as industrial standards. OWL is the successor of

DAML+OIL, which originated from the union of DAML (USA) and OIL (EU) on his part.

In the following we will introduce notations and languages for ontologies with respect to their "Reasoning-ability" and the capability to express requirements. The languages introduced in the sections below range from Topic Maps, which are mostly useful for navigation-based approaches, over logic-based notations like KIF and OWL to case-based approaches.

5.3.1 Topic Maps

The intended goal of Topic Maps [55] is the management of unstructured information. Actually, it can be seen as the implementation of an index catalogue. They store and model meta-data about the knowledge to be managed. Due to their structured management of knowledge, it is possible to implement efficient full-text search algorithms and they provide an easy navigation through the modelled knowledge. Topic Maps are often compared in analogy to books in literature. The content of the book is equivalent to the knowledge to be structured whereas the index catalogue at the end of the book enables an easy navigation through the book by providing information about the content and the relationships between sections. Topic Maps contain building blocks similar to an index catalogue of a book, which contains basically entries (topics) and page numbers (occurrences) at which the topics can be found: (see figure 17):

- Subjects
  Subjects represent the concepts, terms or information of the knowledge to be modelled. A more well-defined but more "general" definition is given by ISO/IEC 13250:2000: „In the most generic sense, a 'subject' is any thing whatsoever, regardless of whether if exists or has any other specific characteristics, about which anything whatsoever may be asserted by any means whatsoever".
- Topics
  Whereas subjects represent information items of the world to be modelled, topics are the respective representatives in the "virtual world". Besides its unique identifier a topic contains also a human readable name that is used for facilitating the communication between human actors.
- Occurrences
  Occurrences of a topic describe the information items that contribute to the topic. In the book associated analogy, occurrences of a topic are the information carrying elements like sentences, formulas, words, etc.
- Associations
  Associations are used to ease the understanding of specific topics by provid-

ing additional context information. An association can be associated with two or more topics as they contain a large variety of context information. For example, an association connected to a topic about the city of Paris could also be connected to a topic that describes the train connections to Paris.

- Classes
  Topics, occurrances, or associations can be described by classes. A class described at the afore mentioned example about the topic about the city of Paris could be the topic about capital cities, which may possess further instances. It is also possible to define an inheritance relationship between classes, for example a "capital city" could be the child of a parent class "city".

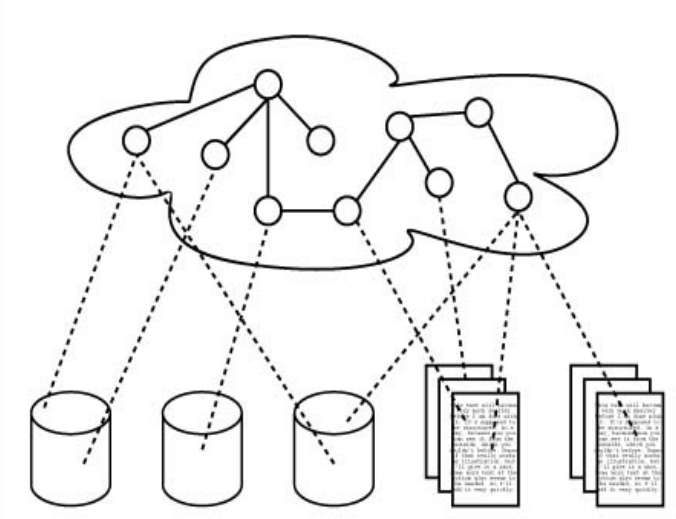


Figure 10 Topic Maps

Topic maps have been at first described in detail in the beginning of the year 2001 by the specification „XML Topic Maps (XTM) 1.0", and the syntax was integrated into the ISO 13250 standard in October of 2001. Today, XTM is a widely used exchange format for topic map tools. In the following, a short example of a typical XTM syntax is given. A more detailed description can be found at the above given reference.

```
<topicMap xmlns="http://www.topicmaps.org/xtm/1.0/"
      xmlns:xlink="http://www.w3.org/1999/xlink">

 <topic id="CapitatID">
  <baseName>
   <baseNameString>Capital</baseNameString>
  </baseName>
```

```
 </topic>

 <topic id="CityID">
  <baseName>
   <baseNameString>City</baseNameString>
  </baseName>
 </topic>

 <topic id="CountryID">
  <baseName>
   <baseNameString>Country</baseNameString>
  </baseName>
 </topic>

<association>
  <instanceOf>
   <topicRef xlink:href="#CountryCityAssoziationID"/>
  </instanceOf>

  <member>
   <roleSpec>
    <topicRef xlink:href="#CountryOf"/>
   </roleSpec>
   <topicRef xlink:href="#CountryID"/>
  </member>

  <member>
   <roleSpec>
    <topicRef xlink:href="#CityOf"/>
   </roleSpec>
   <topicRef xlink:href="#CityID"/>
  </member>
 </association>
 ..
</topicMap>
```

5.3.2 FLogic

FLogic uses a different approach compared to the above introduced Topic Maps. FLogic is an ontology modelling language that stems from object oriented deductive databases. The language provides traditional concepts of object-oriented data structuring models like classes, instances and relationships. Therefore, a well-defined mathematical model theory can be applied to language statements. It is also possible to model classes and instances of classes.

Every class and instance is positioned in a partial order. That's why according to the definition of the logical theory, FLogic has a syntax of the second level but the semantics of the first level. It is possible to define logical inference rules as class methods in FLogic. Therewith, an ontology may not only contain structural information but also the logical information that controls the interaction among objects. Thus, FLogic has a greater power of expressiveness compared to topics maps for example. In the following example, an ontology will be described in the FLogic syntax.

```
%Definition of the class level
person[name => string;              %Attribute name (single value)
      age => int;                   %age (single value)
      friends =>> person;           %Set of friends of the class person (multi
value)
      son =>> person;               %Set of sons (multi value)
      daughter =>> person;  %Set of daughters  (multi value)
      father =>man;         %Father (single value)
      mother =>woman].              %Mother (single value)
man::person.                              %person is a generic class of man
woman::person.                %person is a generic class of woman

%Definition of the instance level
mike:man[name->"Mike";
         age -> 42;]

mary:woman[name -> „Mary";
           age -> 40;
           friends ->> {bob, sally]};].

peter:man [name -> „Peter";
          age -> 15;
          father -> mike;
          mother -> mary;].

thomas:man [name -> „Thomas";
           age -> 11;
           father -> mike;
           mother -> mary;].

%Axioms
FORALL X, Y X[son->>Y] <- Y:man[father -> X].
FORALL X, Y X[son->>Y] <- Y:man[mother -> Y].
FORALL X, Y X[daughter->>Y] <- Y:woman[father -> X].
FORALL X, Y X[daughter->>Y] <- Y:woman[mother -> Y].
...
```

%Query
FORALL X <- mary[son->>X].
...

The above specified ontology consists of four sections. The first section specifies the hierarchy among the different terms. Double colons :: are used to denote inheritance relationships. The first expression defines the signature of the class person: person is a concept on the class level and contains a name (name), an age (age) and a set of friends (friends). The following two expressions define two subclasses of the class person.

The second section of the ontology specifies instances, their relations to other classes and the relations among instances. A colon : is used to separate the instance name and the class name. A major characteristic of FLogic is that the instance level and the class level are not specified separately. This gives the advantage to specify and process instances and classes in a consistent and uniform language. Nevertheless, it is possible to separate relations on the concept level (=> single value, =>> multi value) and on the instance level (-> single value, ->> multi value).

The FLogic syntax is also capable of specifying composite statements, e.g. mary:woman[name -> „Mary"; age -> 40; friends ->> {bob, sally]};]. The statement simultaneously expresses that mary is an instance of the class woman and assigns the attribute values of the instance mary.

The third section specifies the characteristics of the classes by using FLogic axioms, similar to a first level logical syntax. These axioms can be used to define constraints on the class and on the instance level. For example, the axiom FORALL X, Y X[son->>Y] <- Y:man[father -> X] describes the meaning of the son-attribute.

FLogic allows the statement of queries based on the specified models. FLogic uses its own language to specify queries and does not rely on other query languages like SQL. Axioms without a rule head are used to express queries, for example FORALL X <- mary[son->>X] provides all sons of mary (result: X = {peter, thomas}).

5.3.3 KIF

The KIF (Knowledge Interchange Format) is another language used to specify knowledge on information. Many different types of information exist and this poses diverse requirements on languages used to represent the knowledge on the information. For example, SQL and OQL are languages, which are especially useful to describe information stored in databases. Object-oriented languages

are often used for data and method centralized modelling. Therefore, it is difficult to develop one general applicable language that is capable of describing all potential information. The approach used by KIF is, similar to the FLOgic, based on first level logic. Logic is regarded as a common denominator of information sources as it is especially useful to represent „objects“and their relations. The KIF specification describes the syntax of KIF in BNF and defines the semantics in detail. A basic concept of KIF is to describe simple data with arguments. For example, the arguments define the unique identifier of the employee and the salary:

(salary 234-234 34000)

(salary 473-622 37000)

Complex expressions can be described in KIF by combining statements. The following example defines that a specific house is larger than another one:

(> ((width haus1) (length haus1))*

((width haus2) (length haus2)))*

KIF contains many logical operators to associate information with logical relations like negation, aggregation, rules, quantified formulas, etc.. The following example describes a complex expression in KIF, e.g. the even power of a real number is positive:

(=> (and (real-number ?x)

(even-number ?n))

(> (expt ?x ?n) 0))

KIF can also be used to describe meta-knowledge about knowledge. The symbol ' is used to denote the change to the meta-level. The following expression describes: John believes that the moon is made of stilton cheese:

(believes john '(material moon stilton))

The number of meta-levels is unlimited. Thus, it is possible to create any nesting of information required. Each level of information describes statements on information of the underlying meta-level. Furthermore, information on meta-levels can be associated with information on other levels, as the following example describes. Mary believes everything that john believes:

(=> (believes john ?p) (believes mary ?p))

The semantic of KIF is also similar to the first level logic. Besides some extensions like the meta-levels, the KIF fundamentals are based on first level logic characteristics like compactness.

5.3.4 RDF(S)

The Resource Description Framework (RDF) [56] and RDF-Schema (RDFS) is an established standard developed by the World Wide Web Consortium (W3C). RDF was developed to provide machine readable statements on web resources. Web resources are for example web pages or other objects with a uniform resource locator (URI).

RDF provides basic and fundamental concepts for statement exchange, processing (e.g. reasoning) and for the reuse of statements. Simple and unstructured meta-data can be specified and declared with the supplied syntax and semantic. RDF uses three different objects that build up the RDF vocabulary:

Statements are given about *resources.* Each resource can be uniquely identified by its uniform resource identifier (URI).

Literals represent values (e.g. character string), which are used to specify the characteristics of resources.

Properties are defined links between *resources* and *literals*. A property can refer to literals as well as resources.

RDF itself is a application of the XML, i.e. it extends the standard XML model and syntax by specifications for web resources. In the following section, we will introduce the basic specification of meta-data with resources in RDF.

5.3.4.1 RDF Syntax

RDF [56] is based on the idea that every *object* has *properties*, which are described by *value*s. So, every resource can be specified with a statement. A RDF-statement is a triple specified as *RDF-statement := (Subject , Predicate , Object),* whereas a *subject* defines the resource based on the uniform resource identifier, short URI. A *predicate* describes a specific property of the resource and the o*bject* defines a concrete value of the property. The identification of a resource in the web is done with the URI. It is an unique identifier and represents a more general type as a URL (Uniform Resource Locator). The advantage is that it can be automatically processed by a machine.

We will introduce an example of a RDF-statement in the following. The information to be described in RDF is as follows:

„The resource http://www.xyz.de has an author, whose value is, Herr Müller' ."

This information can be represented by the following triple:

Subject (Resource): http://www.xyz.de

Predicate (property): author

Object (value): Herr Müller

The figure below illustrates this statement:

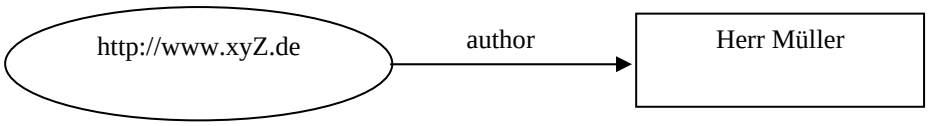


Figure 11 RDF example

Both the written statement and the picture above depict the same following issue:

<subject> has <predicate> <object>

They describe the structure of the RDF document, which is written in XML. The example specified in XML/RDF, which is a XML version extended with RDF statements would be as follows:

```
<?xml version ="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:s="http://description.org/schema/">
<rdf:Description about="http://www.xyz.de">
<s:Author>Herr Müller</s:Author>
</rdf:Description>
</rdf:RDF>
```

This description can be automatically processed and can be used for communications between machines.

RDF also uses the XML concept *namespace*. A standard namespace in RDF is rdf, which refers to the RDF syntax resource:

RDF <xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

It is also possible to add additional namespaces, for example:

<xmlns:s="http://description.org/schema/">

RDF uses these namespaces in order to separate between RDF elements (standard namespace) and property elements of resources.

Another important concept in RDF are *containers*. An element in RDF describes a resource that has several properties of the same type. Container objects are used to specify these elements. RDF provides 3 different types of containers:

Bag: A "bag" describes a set of properties with an arbitrary order.

Seq: A sequence is similar to a bag, with the exception that the order of the properties is of importance.

Alt: Alternatives provide a set of properties of which only one can be chosen.

5.3.4.2 RDF Schema

RDF [57] itself is a completely graph based model and provides no means for structuring information. For this reason RDF Schema (RDFS) was designed as an abstract RDF language. RDFS provides the functionality to define classes, assign instances and define relations among classes. Modelling primitives are themselves defined as relationships with specific properties. For example, the statement expressing the issue that a resource #person represents an abstract class for all persons is defined by the relation rdf:type between #person and rdfs:Class. The concept rdf:type itself is a relation that contains a specific semantic, which is specified outside of the RDF model.

5.3.5 DAML+OIL

DAML+OIL [58] is an ontology language developed by DAML (Darpa Agent Markup Language) and the developers of OIL. DAML+OIL has been submitted to W3C as a basis for the W3C Web Ontology Language OWL. DAML+OIL is, unlike OIL, completely layered on top of RDF(S). To be more specifically, DAML+OIL is an extension of RDF(S) and XML that provides a set of constructs in order to make ontologies machine readable and machine comprehensible. Especially, objects and their relationships can be described and automatically analyzed. This can sometimes pose problems. Some restrictions in DAML+OIL can not be expressed in RDF(S), which means that in some cases decidability is lost (namely when cardinality constraints are applied to transitive properties). In addition, the standardized description enhances the exchange of ontologies.

The DAML+OIL also integrates three major aspects:

- The language uses a formal semantic and provides reasoning support
- The language makes also use of epistemological modelling primitives
- The language contains a standard for XML and RDF based syntactical exchange

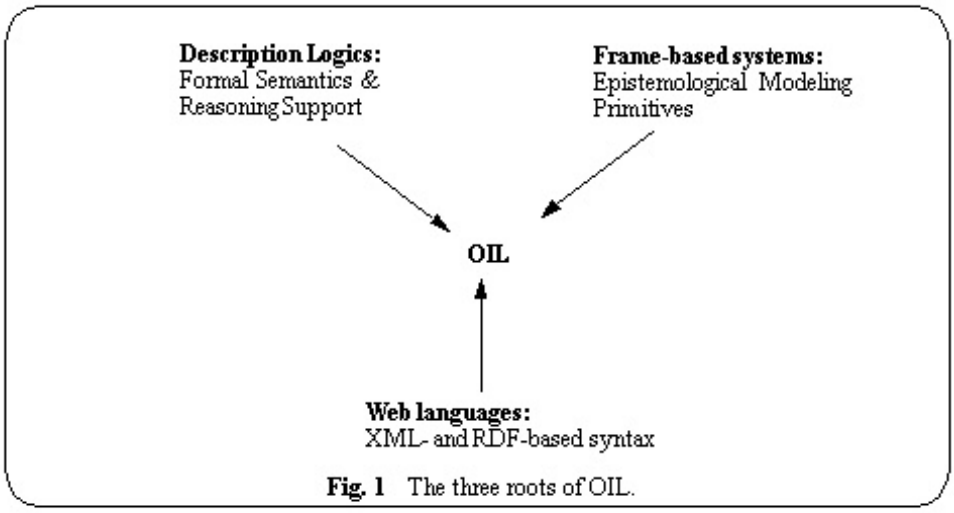


Figure 12 OIL Aspects

DAML+OIL describes an ontology as a three layered set of components:

- An object layer, describing concrete instances of an ontology;
- a meta-layer, describing the definition of the ontology;
- and a second meta-layer or ontology container level , that describes concrete information about characteristics of the ontology.

A major difference between OIL and DAML+OIL is the support for primitive and more complex data types in DAML+OIL, where in OIL only the string data type is supported. The data types used come directly from XML Schema and the domains of interpretation of classes and of data types are completely disjoint. An important reason given here is that adding a logical theory for each data type would lead to an immensely large and complex language.

The differences between OIL and DAML+OIL are, from a language constructs point of view, relatively trivial. For most constructs there is a one-to-one mapping or a simple translation [58]. With respect to OIL, DAML+OIL adds true layering on top of RDF(S) and usage of XML Schema datatypes, making it more of a "true" Semantic Web language than OIL. Currently a large number of DAML+OIL ontologies have been created and the language has been accepted

as the de facto standard for ontologies for the Semantic Web until the arrival of the OWL language. Furthermore, DAML+OIL acts as the basis for the new Web Ontology Language OWL.

5.3.6 OWL

Using the DAML+OIL language as the basis, the W3C Web Ontology working group has created a new ontology language for the Semantic Web, which implements true layering on top of RDF(S) in its current version and incorporates the wishes of many stakeholders, both academic and industrial. OWL [59] actually consists of a set of three dialects, namely OWL Lite, OWL DL, and OWL Full, in a layered approach. This means that OWL Lite is a subset of OWL DL and OWL DL is a subset of OWL Full.

OWL is supposed to be an extension of RDF(S). There are, however, some problems when layering OWL on top of RDF(S). The relationship between RDF(S) and XML is simply syntactic; XML is used as to serialize RDF. However, the relation between OWL and RDF(S) has a big semantic component as well. According to RDF Semantics [56] RDF "sees" the syntactic definition of an ontology and it can draw conclusions that OWL can not; this is mainly because many more constraints and restrictions can be expressed in OWL that cannot be expressed in RDF. Therefore, not every model for an RDF representation is also a model for the OWL ontology. Because of this, OWL model theory cannot be defined as an extension to RDF model theory.

The basic elements of an OWL-ontology are classes, properties, instances of classes and relationships between instances. An instance of a class is also called "thing" or „individual". Properties can be further specified as characteristics like transitive, symmetric, functional, inverseOf, InverseFunctionalProperty. Furthermore, it is possible to define constraints on properties like allValuesFrom, someValuesFrom, Cardinality, hasValue. The following constructs can be used to create complex classes: intersectionOf, unionOf, complementOf; oneOf; disjointWith. Further details of the OWL can be found at [60].

The following example [60] describes the specification of a complex class as an intersection: Burgundy White wine as intersection of wine made of Burgundy and white wine.

```
<owl:Class rdf:ID="WhiteBurgundy">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Burgundy" />
    <owl:Class rdf:about="#WhiteWine" />
  </owl:intersectionOf>
```

```
</owl:Class>
```

In contrast to RDF-Schema, OWL is capable of describing complex relationships between different RDFS classes. Furthermore, precise constraints for classes and properties can be specified:

- Constraints on type and amount-properties
- Specification of 1:1, 1:n and n:m relationships
- Construction of new classes with unions, intersections, or complements of classes

The OWL exists in three different forms: OWL Lite, OWL DL, OWL Full.

OWL Lite supports hierarchies of classifications and simple constraint-features. Especially thesauri and taxonomies can be easily converted in OWL.

OWL DL supports maximal power of expressiveness combines with computational completeness and decidability of reasoning-systems.

OWL Full supports maximal power of expressiveness without any guarantees on computations.

Furthermore, the following rule applies to OWL: A correct OWL Lite ontology is a correct OWL DL ontology. A correct OWL DL ontology is a correct OWL Full ontology. This rule does not apply for reverse direction.

5.3.7 WSMO

The Web Service Modeling Ontology [61] is a meta-ontology which describes relevant aspects for a dynamic composition of web services and how such ontology can be developed. It was initiated by the Digital Enterprise Research Institute (DERI).

The ontology includes the discovery, selection, mediation and invocation of different web services to solve a special problem and is based on the four concepts: web services, ontologies, goals and mediators.

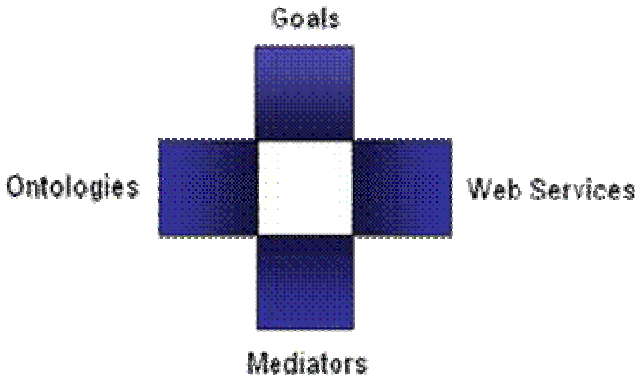


Figure 13 WSMO concepts

In WSMO Ontologies are the key to linking conceptual real world semantics defined and agreed upon by communities of users. An ontology is a formal explicit specification of a shared conceptualization. Ontologies define an agreed common terminology by providing concepts, and relationships between the set of concepts. In order to capture semantic properties of relations and concepts, an ontology generally also provides a set of axioms, which means expressions in some logical framework. The following example describes an example definition of a WSMO ontology:

Class ontology
hasNonFunctionalProperties **type** nonFunctionalProperties
importsOntology **type** ontology
usesMediator **type** ooMediator
hasConcept **type** concept
hasRelation **type** relation
hasFunction **type** function
hasInstance **type** instance
hasAxiom **type** axiom

The goal is the specification of the reason why the client wants to use the web service. It consists of pre-conditions and post-conditions.

Mediators are responsible for the interaction between several web services which are necessary to achieve a special goal. There can be distinguished between different types of mediators: refiners and bridges. Refiners refine an existing component in order to define a new component. Bridges enable the interaction between two components and therefore support reuse.

The web service is a component which realizes functionality. It is responsible for the computation of results sent back to the client or intermediate results sent coordinated by mediators to further web services. WSMO provides service descriptions for describing services that are requested by service requesters, pro-

vided by service providers, and agreed between service providers and requesters. In the following, we describe the common elements of these descriptions as a general service description definition:

Class service
    hasNonFunctionalProperties **type** nonFunctionalProperties
    importsOntology **type** ontology
    usesMediator **type** {ooMediator, wwMediator}
    hasCapability **type** capability multiplicity = single-valued
    hasInterface **type** interface

Further information or more detailed descriptions of WSMO can be found at [61].

5.4 Ontologies and ASG

Ontologies are formal models of a specific domain that support the communication between human and computer based actors. In the case of the ASG platform, that would correspond to the service customers and providers and to the ASG service registry. This facilitates exchange and sharing of knowledge within the ASG usage community. But it requires a negotiation and an agreement between the group of customers and providers on a socio-cultural level regarding the terminology and relationships used to describe the services. To be of any practical use, a well-defined notation is required for ontologies. A general accepted and well-defined language for ontologies is especially important for integrating various data sources and resources and therefore services provided by various partners. Currently, WSMO [61] is used by the other work-components to facilitate the discovery, selection, mediation and invocation of the different services provided by the ASG platform to solve a special customer problem.

6 Conclusion

This survey serves as a basis for the further development of a requirements engineering method for the ASG development methodology. In this survey, we introduced several different types of requirements engineering techniques, which address specific requirements engineering aspects related to the ASG platform. Especially three different levels of requirements became prominent during the analysis of requirements engineering tasks for the ASG platform. The first level is the service customer level that represents the level of the ASG platform at which service requests are raised by service customers. The second level is the service grid level, which represents the level of the ASG platform at which the service discovery and composition takes place based on the raised service request by a customer. And the third level is the service provider level that represents the level of the ASG platform at which service providers semantically specify their provided services and register them to the service registry of the ASG platform so that a service can be discovered and executed to fulfil a customer request.

The approach described in section 2.3.1 to describe Non-functional requirements has been initially developed for embedded systems but shows great adaptation potential for the ASG platform and service-oriented application development. The approach is based on the explicit usage of use cases for describing requirements and on quality models derived from the ISO 9126 standard, as well as general problems and challenges when working with NFR. Use cases are a prominent specification technique for functional behaviour of software products and can also be used for describing business processes of service customers. The approach for NFRs [6] introduced uses use cases as main technique, though the general principle of having a structured and experience-based process is applicable to other techniques as well. Use cases are also already used for describing the usage of the ASG platform and to specify the user scenarios by the C-7 Workcomponent. The usage of an integrated and holistic usage of a uniform and consistent representation form for the various requirements on the three levels seems to be necessary to develop a requirements engineering process for the ASG platform.

BPEL can be used by service providers to provide new services that are based on existing web-services. With the help of BPEL the interaction of different services can be described forming a business process that represents a new service that may be published and invoked.

WSMO [61] is already used by the other workcomponents to address the service grid requirements level in order to facilitate the discovery, selection, media-

tion and invocation of the different services provided by the ASG platform to solve a special customer problem.

Further work will be based upon the introduced and selected techniques mentioned above to develop a requirements engineering approach that makes explicit usage of a common representation form for functional and for non-functional requirements throughout the requirements engineering process. This enables us to create an integrated mapping of the customer related business process and thus specific "service need" to the provided services registered within the ASG registry.

References

[1] Kotonya, G.; Sommerville, I.: Requirements Engineering – Processes and Techniques. John Wiley & Sons, 1997.

[2] Weske, M., et al.: Technical Annex 1 – Adaptive Service Grid, Description of Work. Proposal no. 004617, Integrated Project, Sixth Framework Programme, 2004.

[3] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud. PuLSE: A Methodology to Develop Software Product Lines. In Proceedings of the Fifth ACM SIGSOFT Symposium on Software Reusability (SSR'99), Los Angeles, CA, USA, May 1999.

[4] P. Clements and L. Northrop. Software Product Lines: Practices and Patterns. SEI Series in Software Engineering. Addison-Wesley, 2001.

[5] D. M. Weiss and C. T. R. Lai. Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley, 1999.

[6] Kerkow, D.;Dörr, J.;Paech, B.;Olsson; T.;Koenig, T.; Requirements Engineering for Sociatechnical Systems; Fraunhofer IESE; Internal report; 2004.

[7] Kitchenham B. & Pfleeger S. L. (1996) Software quality: the elusive target. IEEE Software, pp. 12-21.

[8] Menasce, D. A. (2002) Software, Performance or Engineering. Workshop on Software and Performance, 239-242.

[9] Chung, L., Nixon, B. A., Yu, E. & Mylopoulos, J. (2000) Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers.

[10] Loucopoulos, P. & Karakostas, V. (1995) System Requirements Engineering, McGraw-Hill.

[11] Gross, F. & Yu, E. (2001) Evolving system architecture to meet changing business goals: an agent and goal-oriented approach. International Conference on Software Engineering-Workshop STRAW.

[12] Shaw, M., Garlan, D. (1996) Software Architecture – Perspectives on an emerging discipline. Prentice Hall.

[13] Abowd, G., Bass, L., Clements, P., Kazman, R., Northrop, L. & Zaremski, A. (1997). Recommended Best Industrial Practice for Software Architecture Evaluation. CMU/SEI-96-TR-025, Software Engineering Institute, Carnegie Mellon University.

[14] Barbacci, M. R., Klein, M. H. & Weinstock, C. B. (1997) Principles for Evaluating the Quality Attributes of a Software Architecture. CMU/SEI-96-TR-036, Software Engineering Institute, Carnegie Mellon University.

[15] Bass, L., Clements, P. & Kazman, R. (1998) Software Architecture in Practice. Addison-Wesley.

[16] Sutcliffe, A. & Minocha, S. (1998) Scenario-based Analysis of Non-Functional Requirements. Workshop on Requirements Engineering For Software Quality.

[17] Sindre, G. & Opdahl, A. (2000) Eliciting Security Requirements by Misuse Cases, Proc. TOOLS Pacific 2000, pp. 120-131.

[18] Alexander, I. (2001) Misuse Case Help To Elicit Nonfunctional Requirements. IEE CCEJ.

[19] Petriu, D. & Woodside, M. (2002) Analysing Software Requirements Specifications for Performance. Workshop on Software and Performance, pp. 1-9.

[20] Firesmith, D. (2003) Security Use Cases. Journal of Object Technology, 2 (3), 53-64.

[21] Clements, P., Bass, L., Kazman, R. & Abowd, G. (1995) Predicting Software Quality by Architecture-Level Evaluation. Proceeding of the Fifth International Conference on Software Quality.

[22] Liu, L. & Yu, E. (2001) From requirements to architectural design – using goals and scenarios. International Conference on Software Engineering - Workshop, STRAW.

[23] Egyed, A., Grünbacher, P. & Medvidovic, N. (2001) Refinement and evolution issues in bridging requirements and architecture – the CBSP approach. International Conference on Software Engineering-Workshop STRAW

[24] In, H., Kazman, R. & Olson, D. (2001) From requirements negotiation to software architectural decisions, International Conference on Software Engineering -Workshop STRAW.

[25] Cysneiros, L. N. & Leite, J. C. S. P. (2001) Driving Non-Functional Requirements to Use Cases and Scenarios. XV Brazilian Symposium on Software Engineering.

[26] Cockburn A. (2001) Writing Effective Use Cases. Addison Wesley.

[27] Kazman, R., Klein, M. & Clements, P. (1999) ATAM: Method for Architecture Evaluation. CMU/SEI-2000-TR-004, Software Engineering Institute, Carnegie Mellon University.

[28] Kazman, R., Bass, L., Abowd, G. & Webb, M. (1994) SAAM: A Method for Analyzing the Properties of Software Architectures. Proceedings of the 16th International Conference on Software Engineering, pp. 81-90.

[29] Kazman, R., Abowd, G., Bass, L. & Clements, P. (1996) Scenario-Based Analysis of Software Architecture. IEEE Software.

[30] Moreira, A., Brito, I. & Araújo, J. (2002) A Requirements Model for Quality Attributes, Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, International Conference on Aspect-Oriented Software Development, University of Twente, Enschede, Holland.

[31] Paech, B., Dutoit, A., Kerkow, D. & von Knethen, A. (2002) Functional requirements, non-functional requirements and architecture specification cannot be separated – A position paper. International workshop on Requirements Engineering for Software Quality.

[32] Paech, B., von Knethen, A., Doerr, J., Bayer, J., Kerkow, D., Kolb, R., Trendowicz, A., Punter, T. & Dutoit, A. (2003) An experience based approach for integrating architecture and requirements engineering. International Conference on Software Engineering -workshop STRAW.

[33] Basili, V. R. & Rombach, H. D. (1988) The TAME project: Towards improvement-oriented software environments. IEEE Transactions on Software Engineering, vol. 14, no. 6, pp. 758-773.

[34] Basili, V.R. (1992) Software Modeling and Measurement. The Goal/Question/Metric Paradigm. Computer Science Technical Report Series NR: CS-TR-2956 / NR: UMIACS-TR-92-96.

[35] Klein, M. & Kazman, R. (1999) Attribute-based Architectural Styles. CMU/SEI-99-TR-022, Software Engineering Institute, Carnegie Mellon University.

[36] IEEE Recommended Practice for Software Requirements Specifications, IEEE Std. 830-1998

[37] C. Denger, B. Paech, S. Benz, „Guidelines - Creating Use Cases for Embedded Systems", IESE-Report 078.03/E, Kaiserslautern, 2003.

[38] S. Lauesen, Software Requirements. Styles and Techniques. Addison-Wesley, 2002.

[39] I. Alexander, "Misuse cases: use cases with hostile intent", IEEE Software, 20(1), 58-66, 2003.

[40] D. Harel, M. Politi, Modeling Reactive Systems with Statecharts. The Statemate Approach, McGraw-Hill, 1998.

[41] C. Heitmeyer, A. Bull, C. Gasarch, B. Labaw, "SCR: a toolset for specifying and analysing requirements", Conference on Computer Assurance (COMPASS), 1995.

[42] M. Fowler, UML Distilled. A Brief Guide to the Standard Object Modeling Language. Addison-Wesley, 2003.

[43] Message Sequence Charts (MSC), ITU-T Standard Z.120.International Telecommunication Union. 1996.

[44] S.J. Prowell, J.H. Poore, "Foundations of sequence-based software specification", IEEE Transaction on Software Engineering, 29(5), 417-429, 2003.

[45] IDS Scheer, http://www.ids-scheer.de/

[46] Andrews T. et al; Business Process Execution Language for Web Services Version 1.1; 2003; http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[47] Leymann F.; Web Services Flow Language (WSFL); 2001; http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[48] Technology Report; XLANG; 2001; http://xml.coverpages.org/xlang.html

[49] Chinicci et al; Web Services Description Language (WSDL) Version 1.2; 2003; http://www.w3.org/TR/2003/WD-wsdl12-20030611/

[50] Business Process Management Initiative; www.bpmi.org

[51] Business Process Modeling Notation; http://www.bpmi.org/bpmn-spec.htm

[52] Intalio n3 Server; http://www.intalio.com/products/server/index.xpg

[53] Intalio ZeroCode Initiative;
http://www.intalio.com/education/zerocode/index.xpg

[54] T. Gruber: A Translation Approach to Portable Ontology Specifications. In: Knowledge Acquistion, Band 5, Seiten 199–220, 1993.

[55] http://www.topicmaps.org/xtm/

[56] http://www.w3.org/TR/rdf-syntax-grammar/

[57] http://www.w3.org/TR/rdf-schema/

[58] http://www.daml.org/

[59] http://www.w3.org/2004/OWL/

[60] http://www.w3.org/TR/2004/REC-owl-guide-20040210/

[61] http://www.wsmo.org/TR/d2/v1.1/

Document Information

Title:	Requirements Specification Survey Adaptive Services Grid Deliverable D6.I-1
Date:	February 28, 2005
Report:	IESE-133.05/E
Status:	Final
Distribution:	Public