Grant Agreement N° 215483

Title:	*Quality Reference Model for SBA*
Authors:	*CITY, CNR, FBK, INRIA, Lero, POLIMI, SZTAKI, TUW, UniDue, UPM, UStutt, Tilburg*
Editors:	*Andreas Gehlert (UniDue), Andreas Metzger (UniDue)*
Reviewers:	*Michael Parkin (Tilburg)*
	Barbara Pernici (POLIMI)
Identifier:	*Deliverable #CD-JRA-1.3.2*
Type:	*Contractual Deliverable*
Version:	*1.0*
Date:	*2008-03-16*
Status:	*Final*
Class:	*External*

Management Summary

The aim of this deliverable is two-fold. Firstly, it depicts the research vision of the workpackage, including the research challenges that will be addressed by the S-Cube consortium. Secondly, the deliverable aims at defining the S-Cube quality reference model. This reference model is intended to provide the S-Cube consortium with a unified terminology for describing different quality attributes of service-based applications. To this end, important quality models from service-oriented computing, business process management, grid computing and software engineering are analyzed. The quality attributes which are defined in these models and which are relevant for S-Cube are extracted and synthesized into the S-Cube quality reference model.

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n° 215483 (S-Cube).

Members of the S-Cube consortium:

University of Duisburg-Essen (Coordinator)	Germany
Tilburg University	Netherlands
City University London	U.K.
Consiglio Nazionale delle Ricerche	Italy
Center for Scientific and Technological Research	Italy
The French National Institute for Research in Computer Science and Control	France
Lero - The Irish Software Engineering Research Centre	Ireland
Politecnico di Milano	Italy
MTA SZTAKI – Computer and Automation Research Institute	Hungary
Vienna University of Technology	Austria
Université Claude Bernard Lyon	France
University of Crete	Greece
Universidad Politécnica de Madrid	Spain
University of Stuttgart	Germany
University of Hamburg	Germany
Vrije Universiteit Amsterdam	Netherlands

Published S-Cube documents

All public S-Cube deliverables are available from the S-Cube Web Portal at the following URL:

http://www.s-cube-network.eu/results/deliverables/

The S-Cube Deliverable Series

Vision and Objectives of S-Cube

The Software Services and Systems Network (S-Cube) will establish a unified, multidisciplinary, vibrant research community which will enable Europe to lead the software-services revolution, helping shape the software-service based Internet which is the backbone of our future interactive society.

By integrating diverse research communities, S-Cube intends to achieve world-wide scientific excellence in a field that is critical for European competitiveness. S-Cube will accomplish its aims by meeting the following objectives:

- Re-aligning, re-shaping and integrating research agendas of key European players from diverse research areas and by synthesizing and integrating diversified knowledge, thereby establishing a long-lasting foundation for steering research and for achieving innovation at the highest level.
- Inaugurating a Europe-wide common program of education and training for researchers and industry thereby creating a common culture that will have a profound impact on the future of the field.
- Establishing a pro-active mobility plan to enable cross-fertilisation and thereby fostering the integration of research communities and the establishment of a common software services research culture.
- Establishing trust relationships with industry via European Technology Platforms (specifically NESSI) to achieve a catalytic effect in shaping European research, strengthening industrial competitiveness and addressing main societal challenges.
- Defining a broader research vision and perspective that will shape the software-service based Internet of the future and will accelerate economic growth and improve the living conditions of European citizens.

S-Cube will produce an integrated research community of international reputation and acclaim that will help define the future shape of the field of software services which is of critical for European competitiveness. S-Cube will provide service engineering methodologies which facilitate the development, deployment and adjustment of sophisticated hybrid service-based systems that cannot be addressed with today's limited software engineering approaches. S-Cube will further introduce an advanced training program for researchers and practitioners. Finally, S-Cube intends to bring strategic added value to European industry by using industry best-practice models and by implementing research results into pilot business cases and prototype systems.

S-Cube materials are available from URL: http://www.s-cube-network.eu/

Foreword

The aim of this deliverable is two-fold.

- Firstly, it provides a summary of the research vision of the workpackage, including an overview of research objectives that will be addressed by the S-Cube consortium.
- Secondly, the deliverable aims at defining the S-Cube quality reference model (QRM). This reference model is intended to provide the S-Cube consortium with a unified terminology for describing different quality attributes of service-based applications. To this end, important quality models from service-oriented computing, business process management, grid computing and software engineering are analyzed. The quality attributes which are defined in these models and which are relevant for S-Cube are extracted and synthesized into the S-Cube quality reference model.

Acknowledgments: The editors would like to thank the team from Politecnico di Milano (especially Elisabetta di Nitto) for organizing the JRA-1 workshop in Milano and the team from Vrije Universiteit Amsterdam (especially Patricia Lago) for organizing the workshop of the Integration Committee in Amsterdam. Those meetings have provided us with the opportunity to discuss significant elements of the deliverable. In addition, we thank all S-Cube members who have contributed to this deliverable and especially Michael Parkin and Barbara Pernici for their helpful and detailed comments on earlier versions of the document.

Contents

Chapter 1

Workpackage Vision

As described in S-Cube's Description of Work (DoW), the general research goal of workpackage WP-JRA-1.3 is to define novel principles, techniques and methods for defining, negotiating and assuring end-to-end quality across the functional layers as well as across networks of service providers and consumers. This chapter provides a summary of the research vision for this workpackage, which refines that general research goal. To devise this vision, the WP members have analyzed the existing problems and gaps in the state of the art, which has been surveyed in the initial deliverable of this workpackage [110] as well as in S-Cube publications [39, 109]. The vision has been jointly defined and agreed on during several meetings, including – besides others – a JRA-1 research workshop, a meeting of the S-Cube Integration Committee and and a meeting of the research activity and workpackage leaders.

Motivation

Services are more and more provisioned in the context of short-term, volatile and thus highly dynamic (business) relationships and (business) processes [110]. Those relationships involve service providers which are not known during the design time of the service-based application. Thus, different from traditional software systems, service-based applications require the composition and coordination of services within highly distributed environments, cutting across the administrative boundaries of various organizations.

To guarantee the desired end-to-end quality of those service-based applications, contracts between the service providers and the service requestors (also known as service consumers) on quality aspects of services have to be established [34]. In general, a contract is a formal agreement between two or more parties to create mutual business relations or legal obligations. Contracts can have different parts, such as the definition of business partners, the specification of functional obligations, and quality, price, and penalties related with the object of the agreement. Workpackage WP-JRA-1.3 in particular focuses on quality contracts, or more general on those parts of Service Level Agreements (SLAs) which deal with statements about the services quality levels on which the service requestor and the providers have reached an agreement. Other aspects of the contracts, such as parties' identification, legal obligations, or penalties for contract violation, which are also aspects covered in SLAs or general contracts, will not be in the focus of this workpackage.

Key Research Objectives

Based on the general life-cycle of electronic contracts [116, 129], three main activities relevant for quality contracts within service-based applications can be identified: quality definition, quality negotiation and quality assurance. Within each of these activities, key problems and key research objectives have been identified. Below, a summary of these research objectives in the context of the quality activities is provided (see Figure 1.1 for an overview):

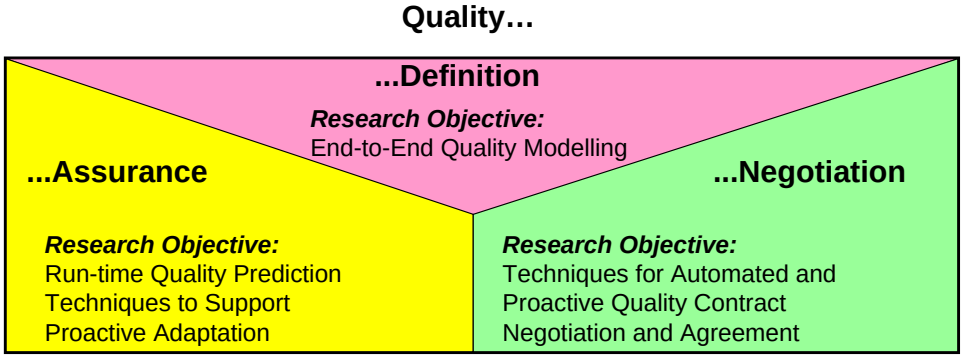

Figure 1.1: Activities relevant for quality of service-based applications and key research objectives of workpackage

- *Quality definition*: In electronic contracting, the contract definition activity concerns the establishment of a model or language for the definition of contract terms, which is understood and shared by the contracting parties. This model or language then is used to instantiate an actual contract (e.g., a SLA) that reflects the domain dependent interests of providers and consumers.

  For what concerns establishing end-to-end quality, we currently miss an understanding on how to aggregate quality levels stipulated in individual quality contracts (e.g., as part of SLAs) across layers and across networks of service providers and consumers. As a first step to address this problem, this workpackage aims at defining a comprehensive and integrated quality reference model, including quality attributes which are relevant at the different layers of a service-based application. To this end, the quality reference model incorporates and aligns the quality attributes relevant for WP-JRA-2.1 (BPM), WP-JRA-2.2 (service composition) and WP-JRA-2.3 (service infrastructure) as well as quality attributes relevant for software engineering.

  As a second step to address the problem of establishing end-to-end quality, this workpackage sets out to devise an extensible and rich quality definition language. As existing languages for quality definition offer limited capabilities for automated negotiation and quality-aware service composition, this language will provide means to precisely express different levels of quality as well as their dependencies. The quality definition language will be based on the quality attributes as identified in the quality reference model.

  Together, the quality reference model and the quality definition language will enable the *modelling of end-to-end quality for service-based applications*. The model and language thus integrate knowledge from the S-Cube disciplines business process management, service-oriented computing, grid computing and software engineering such that this knowledge can be exploited for end-to-end quality definition, negotiation and assurance. As an example, based on layer-specific monitoring data (see WP-JRA-1.2 and WP-JRA-2.1 – WP-JRA-2.3), techniques will be devised, which allow aggregating those isolated quality levels into end-to-end quality levels that can be checked against end-to-end quality requirements.

  Besides others, the quality reference model and the quality definition language serve as input for requirements engineering techniques in WP-JRA-1.1 and monitoring techniques and mechanisms in WP-JRA-1.2. Work on end-to-end quality modelling is pursued in task T-JRA-1.3.1 'Quality Reference Model for Service-based Applications', as well as in task T-JRA-1.3.2 'Specifying and Negotiating End-to-End Quality and SLAs'.

- *Quality negotiation*: Establishment of an electronic contract concerns the set of tasks required for defining an actual contract (e.g., SLA) based on the model or language for the definition of contract terms (see above). This may involve the selection of the service provider (the contract

partner) among a set of potential providers, the negotiation of the contract terms between the selected provider and the service consumer, and the agreement to the contract terms.

Automatic quality negotiation and agreement become important as, in dynamic service settings, more and more decisions must be taken by the service-based application during run-time. In addition, proactive negotiation and agreement is a key prerequisite for effective run-time quality contract negotiation, since negotiation does imply significant computational costs. Proactive negotiation is based on predicting the need for (re-)negotiation during run-time. Thus, this workpackage focuses on *novel quality negotiation and agreement techniques for automatically and proactively establishing contracts on quality attributes*.

Exploiting codified human computer interaction and context knowledge will be key to supporting the envisioned quality negotiation techniques. As an example, we plan to exploit user and task models from WP-JRA-1.1, which codify (i.e., make machine-accessible) user preferences and characteristics, to devise service negotiators that perform the negotiation process on behalf of the service consumers and providers. For pro-active negotiation, we expect synergies with the quality prediction techniques for pro-active adaptation (see below). Work on automated and proactive quality negotiation is pursued in task T-JRA-1.3.2 'Specifying and Negotiating End-to-End Quality and SLAs'.

- *Quality assurance*: Contract enactment in electronic contracting concerns tasks for assuring the satisfaction of the contracts. In the case of quality contracts, this implies assuring that the quality levels negotiated and agreed between the service provider and the service requestor are met.

  Currently, assuring that contracts are met is typically performed by monitoring the service-based application. If a deviation is observed, the application then is (reactively) adapted. However, proactive adaptation would offer significant benefits compared to such a reactive adaptation (like not having to compensate for the deviation). Unfortunately, currently we are lacking quality assurance techniques for predicting quality levels and thus triggering proactive adaptation. Thus, this workpackage will elaborate *novel run-time techniques for predicting quality attributes*. These will be contributed to WP-JRA-1.2 as a key means to support pro-active adaptation of service-based applications. In addition, if quality assurance has uncovered a contract violation, this could also mean that the contracts have to be re-negotiated (see above).

  Local quality assurance mechanisms and techniques of the individual layers (i.e., from WP-JRA-2.1, WP-JRA-2.3 and WP-JRA-2.3) as well as techniques from software engineering constitute building-blocks and input for the novel quality assurance techniques devised in this workpackage. Specifically, these techniques will be augmented with run-time information, such that these become applicable during the operation of a service-based application. As examples, we will investigate how existing testing or model analysis techniques have to be extended such that these can be exploited during the operation of the service-based application. Based on these techniques,we will devise prediction techniques for quality attributes to enable pro-active adaptation. For instance, correctness or performance could be predicted by building on techniques similar to online testing [59] or run-time model analysis [49]. Work on run-time quality prediction is pursued in task T-JRA-1.3.3 'Assuring and Monitoring End-to-End Quality Provision and SLA Conformance'

Summarizing, WP-JRA-1.3 will pursue integrative and innovative research to devise novel principles, techniques and methods for defining, negotiating and assuring end-to-end quality across the functional layers as well as across networks of service providers and consumers.

Chapter 2

Building the S-Cube Quality Reference Model

2.1 Motivation

As has been discussed in the previous Chapter, the dynamic agreement of quality contracts and the assurance of those quality contracts becomes a key issue in service-based applications. This not only requires that quality aspects are negotiated and agreed, but also that those are assured during run-time in order to determine whether there is a need for adapting the service-based application or for re-negotiating the quality contracts.

As a prerequisite to quality negotiation and assurance, quality attributes, quality levels and values need to be specified and documented. As summarized above, this requires an integrated and consolidated view across all layers of a service-based application as well as across the chain / network of service providers and consumers. In a service-based application, different kinds of quality attributes (aka. quality characteristics or quality properties) need to be considered and are relevant at the different layers of the service-based application.

As observed in [110, 39, 109], we currently lack techniques and methods that address those quality attributes in a comprehensive and cross-cutting fashion across all layers of a service-based application and integrating all phases of quality contract definition, negotiation, assurance and refinement. In addition, only few approaches consider the context of a service-based application and its impact on quality.

Due to the dynamism of the environment in which service-based applications operate, techniques are needed to aggregate individual quality levels of the services involved in a service composition (or even service-based application) in order to determine and thus assure the end-to-end quality during run-time. This aggregation will typically span different layers of a service-based application and thus a common understanding within and across these layers on what the different quality attributes mean is needed.

To this end, this deliverable sets out to identify relevant quality attributes at the various layers and defined by the different S-Cube research communities (Service-oriented Computing (SOC), Software Engineering (SE), Business Process Management (BPM) and Grid Computing), and to consolidate these into the S-Cube Quality Reference Model (QRM). The S-Cube quality reference model will provide an understanding of which quality attributes are relevant, how they are defined as well as how they relate to each other. Thereby, this quality model explicitly documents and consolidates and aligns the knowledge of the diverse research disciplines in form of a quality taxonomy.

The S-Cube Quality Reference Model will serve as a foundation for further work in this workpackage. As an example, the forthcoming deliverable CD-JRA-1.3.3 'Initial concepts for specifying end-to-end quality characteristics and negotiating SLAs' will set out to define a quality definition language to be used during quality negotiation and assurance. In addition, the quality model will serve as a reference for other S-Cube workpackages; e.g., the quality attributes and their definitions are provided to the S-Cube Knowledge Model (WP-IA-1.1) and the quality reference model will serve as input for require-

ments engineering techniques and processes in WP-JRA-1.1, as well as for monitoring techniques and mechanisms in WP-JRA-1.2.

2.2 Research Method

The S-Cube quality reference model as presented in this deliverable has been developed in three major phases:

1. *Data collection:* Each workpackage partner was asked to select the most important quality models (including potential SLA constituents) in his field and to describe these quality models. One main selection criterion for these quality models was the widespread use of the quality models in academia and industry. Evidence for this usage could be provided by, for example, standardization, importance for stating quality aspects of SLAs, relevant industrial use cases, or by being published in highly ranked publications.

   Chapter 3 describes the relevant quality models coming from the different S-Cube research disciplines (see Section 2.3 for an overview). The quality models analysed have been described with respect to the following four major aspects, according to which each quality model description has been structured:

   - *Introduction:* This aspect provides the motivation and sketches the usage of the respective quality model.
   - *Description of the Model:* This aspect describes the quality model in more detail, and especially includes a list of quality attributes (terms and definitions) together with the context in which they are defined.
   - *Proposed Structure of the Quality Attributes:* If applicable, this aspect explains the structure of the quality attributes as proposed by the respective quality model.
   - *Relevance for S-Cube* This aspect discussed the potential relevance of each particular quality model for quality definition, negotiation and assurance respectively.

2. *Quality attributes analysis:* The aim of this analysis phase is to identify from the lists of quality attributes described in phase 1 (Data collection) those quality attributes that are relevant for services and service-based applications. In addition, synonyms and homonyms in quality attributes coming from different S-Cube research areas and disciplines are identified and resolved.

3. *Consolidating the quality models:* The last phase aimed at consolidating the various quality models. To this end, two kinds of inputs have been taken into account: Firstly, the quality models described in the data collection phase, and secondly, the relevant quality attributes provided in phase 2 (quality attributes analysis). Specifically, quality attributes associated with context and HCI factors, such as usability, accessibility and system-level performance, were taken into account and were related with quality attributes intrinsic to individual services, such as accuracy, availability and performance. As a result of this step, the S-Cube Quality Reference Model (QRM) has been constructed.

   Chapter 4 describes the S-Cube QRM by providing a classification of the relevant quality attributes into quality categories, defining the individual quality attributes and providing a taxonomy of these quality categories and attributes (in the form of a graphical representation).

Notes: According to our initial planning (in the Description of Work), it was planned to take into account the body of knowledge on SLAs in the telecommunications field. However, at this stage in the project, it was considered to be important to integrate the body of knowledge of the four key research

disciplines of S-Cube first, i.e. to align the understanding of quality attributes in software engineering, business process management, service-oriented computing and grid computing.

In addition, parts of this deliverable were intended to analyse SLA properties and situations to be monitored and to provide a taxonomy of requirements for monitorability in SLA. In order to better align the work in WP-JRA-1.3 with that in WP-JRA-1.2, it was agreed to discuss that taxonomy together with the more comprehensive taxonomy of adaptation and monitoring in deliverable CD-JRA-1.2.2 "Taxonomy of adaptation principles and mechanisms".

2.3 Research Disciplines Involved and their Quality Models

The S-Cube QRM synthesizes the body of knowledge on quality modelling of the different S-Cube research disciplines (see Chapter 3 for more details). The body of knowledge of the following disciplines has been considered:

- *Software Engineering:* As a model, which covers general software quality aspects, the ISO software quality model is considered in Section 3.1. More concrete and detailed models used in software engineering are UML-based quality models, which are discussed in Section 3.2. These quality models were selected because the UML is the de-facto standard in the software engineering discipline and it is, therefore, expected that the quality models associated with this standard will be widely recognized and used.
- *Software Engineering / Service-oriented Computing:* A very specific look at quality attributes relevant in software engineering is provided in Section 3.3 by examining QoS attributes that can be statically inferred from development artifacts. Those attributes are relevant for both traditional software as well as for service implementations and thus also cover the SOC discipline. Additionally design by contract models are examined in Section 3.4 as quality models which are relevant both for software engineering and service-oriented computing.
- *Service-oriented Computing:* As specific quality models in SOC, functional quality of service composition models are analyzed in Section 3.5.
- *Business Process Management:* Section 3.6 covers BPM-related quality models for service networks including key performance indicators (KPIs).
- *Grid computing:* Quality models used in Grid computing are covered in Section 3.7.

2.4 Key Terms Used in the Deliverable

Table 2.1 provides the definition for key, generic terms used throughout the deliverable. A detailed list of quality attributes can be found as part of the description of the S-Cube QRM in Section 4.

Term	Definition
Quality	Quality is the degree to which a set of characteristics described as *quality attribute* fulfills a set of requirements. At least three different views on quality can be distinguished: *process quality*, *product quality* and *quality in use*
Process Quality	Process quality is the *quality* of the production process of a product.
Product Quality	Product quality refers to the degree to which a product fulfills its requirements.
Quality in Use	The *quality* of a product evaluated in specific usage contexts and for specific tasks.
Quality Model	A quality model is a model, which structures a set of *quality attributes*.

Term	Definition
Quality Attribute	A quality attribute is used to describe some aspect of *quality*. The term *quality characteristic* is used synonymously.
Quality Characteristic	See *Quality attribute*.

Chapter 3

Analysis of Existing Quality Models

3.1 ISO Software Quality Model (Software Engineering)

ISO-9126 "Software Engineering – Product Quality" [70] is an international standard for measuring and evaluating software quality. The ISO-9126 quality model was chosen because it is the standard for documenting and assuring software quality and has gained practical and theoretical importance in the software engineering fields.

The model consists of four parts (quality model, external metrics, internal metrics and quality in use metrics). For the purpose of this deliverable we will investigate part one, which is a quality model that includes quality characteristics. The quality model is embedded in a framework describing the following four aspects of software quality:

- *process quality:* quality of the production process of a product.
- *internal (product) quality:* degree to which a product fulfils the requirements of the developers.
- *external (product) quality:* degree to which a product fulfils the requirements of the users.
- *quality in use:* product quality evaluated in specific usage contexts and for specific tasks.

The relationship of these quality aspects is depicted in Figure 3.1.

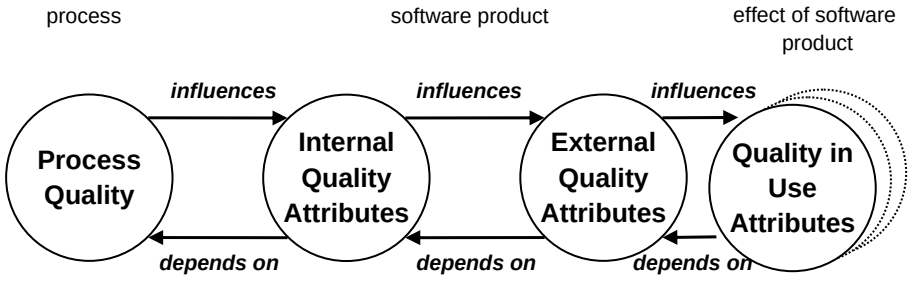

Figure 3.1: Quality Model Framework [70]

The model is intended to be used for [70, p. 1]:

- validating the completeness of a requirements definition,
- identifying software requirements,
- identifying software design objectives,

- identifying software testing objectives,
- identifying quality assurance criteria,
- identifying acceptance criteria for a completed software product.

Description of the Model

The quality model described in ISO-9126 has two parts. The first part covers internal and external quality describing six main quality attributes and their sub-attributes. The six main quality attributes are [70, p. 7]:

- *Functionality*: Functionality is the capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.
- *Reliability*: Reliability is the capability of the software product to maintain a specified level of performance when used under specified conditions.
- *Usability*: Usability is the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions.
- *Efficiency*: Efficiency is the capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.
- *Maintainability*: Maintainability is the capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications.
- *Portability*: Portability is the capability of the software product to be transferred from one environment to another.

The second part of the document specifies four quality attributes for the quality in use aspect. These attributes are [70, p. 12]:

- *Effectiveness*: The effectiveness of a software product describes its capability to enable the user to achieve his or her gaols accurately and completely.
- *Productivity*: The productivity of a software product describes its capability to enable the user to expend an appropriate amount of resources to achieve a given effectiveness.
- *Safety*: Safety is the capability of a software product to achieve an acceptable level of risk to harm people, businesses, other software products or the environment.
- *Satisfaction* Satisfaction is the capability of a software product to satisfy users.

All quality attributes are summarised in Tables 3.1 and 3.2.

Context	Quality Attribute	Definition
Functionality	Suitability	Provide an appropriate set of functions for specified tasks and user objectives.
	Accuracy	Provide the right or agreed results or effects with the needed degree of precision.
	Interoperability	Interact with one or more specified systems.
	Security	Protect information and data so that unauthorised persons or systems cannot read or modify them and authorised persons or systems are not denied access to them.

Context	Quality Attribute	Definition
	Functionality compliance	Adhere to standards, conventions or regulations in laws and similar prescriptions relating to functionality.
Reliability	Maturity	Avoid failure as a result of faults in the software.
	Fault tolerance	Maintain a specified level of performance in cases of software faults or of infringement of specified interfaces.
	Recoverability	Re-establish a specified level of performance and recover the data directly affected in the case of a failure.
	Reliability compliance	Adhere to standards, conventions or regulations relating to reliability.
Usability	Understandability	Enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use.
	Learnability	Enable the user to learn the software's application.
	Operability	Enable the user to operate and control the software.
	Attractiveness	Be attractive to the user.
	Usability compliance	Adhere to standards, conventions, style guides or regulations relating to usability.
Efficiency	Time behaviour	Provide appropriate response and processing times and throughput rates when performing the function, under stated conditions.
	Resource utilisation	Use appropriate amounts and types of resources when the software performs its function under stated conditions.
	Efficiency compliance	Adhere to standards or conventions relating to effiency.
Maintainability	Analysability	Be diagnosed for defiencies or causes of failures in the software, of for the parts to be modified to be identified.
	Changeability	Enable a specified modification to be implemented.
	Stability	Avoid unexpected effects from modifications of the software.
	Testability	Enable modified software to be validated.
	Maintainability compliance	Adhere to standards or conventions relating to maintainability.
Portability	Adaptability	Be adapted for different specified environments without applying actions or means other than those provided for this purpose for the software considered.
	Installability	Be installed in a specific environment.
	Co-existence	Co-exist with other independent software in a common environment sharing common resources.
	Replaceability	Be used in place of another specified software product for the same purpose in the same environment.
	Portability compliance	Adhere to standards or conventions relating to portability.

Table 3.1: Quality characteristics for external and internal software qualities in ISO-9126 [70, p. 7–11)]

Proposed Structure of the Quality Attributes

The quality attributes of the ISO-9126 standard are organised in a tree (as shown in Figure 3.2 on page 13). Further interactions of the quality attributes are not specified by the standard.

Context	Quality Attribute	Definition
Quality in use	Effectiveness	Enable users to achieve specified goals with accuracy and completeness in a specified context of use.
	Productivity	Enable users to expend appropriate amounts of resources in relation to the effectiveness achieved in a specified context of use.
	Safety	Achieve acceptable levels of risk of harm to people, business, software, property or the environment in a specified context of use.
	Satisfaction	Satisfy users in a specified context of use.

Table 3.2: Characteristics for Quality in Use in ISO-9126 [70, p. 12–13]

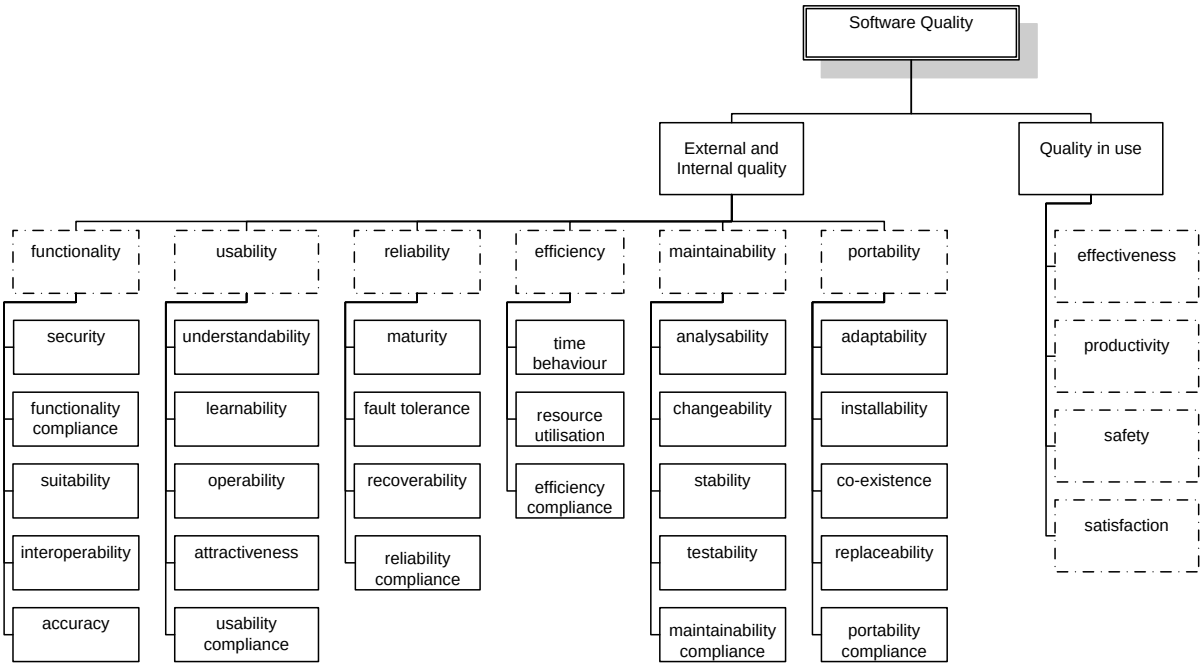

Figure 3.2: Structure of Quality Model and Quality Attributes Based on [70]

Relevance for S-Cube

The ISO standard is relevant for S-Cube as it defines different views on software quality as well as key quality attributes of software systems. The main use of the standard is for identifying important quality attributes and, therefore, initiating the quality assurance for software. The remaining parts of the standard also specify metrics how the quality attributes should be measured (see also Table 3.3).

Goal	Relevance	Rationale/Explanation
Quality Definition	+	The main focus of this quality model is to provide means for defining quality attributes for software systems.
Quality Negotiation	-	There are no techniques to negotiate quality.
Quality Assurance	o	Specification of metrics to measure quality attributes.
Discipline(s)	Software Engineering	

Table 3.3: Potential Relevance of the ISO Standard for the Goals of WP-JRA-1.3 (+ = fully relevant, o = partially relevant, - = not relevant)

3.2 UML-Based Quality Models (Software Engineering)

As the Unified Modelling Language (UML) is the de-facto standard for modelling software systems, we investigate quality models related to the UML in this section. Documenting quality characteristics in UML is mainly achieved by different UML profiles. In the following we discuss the documentation of quality characteristics in standard UML, the UML profile for reliability (Section 3.2.1), the UML Profile for Schedulability, Performance, and Time Specification (SPT; Section 3.2.2) and the UML Quality of Service profile (Section 3.2.3). Some UML profiles such as the Quality of Service Modelling language [46, 47] and the UML Testing Profile [101] define a language extension to describe quality attributes in the UML. Since we are interested in concrete quality models and not in languages for specifying quality here, we exclude these profiles from our analysis.

Quality attributes, especially time constraints can be modelled in standard UML [103, 99]. For example, timing and sequence diagrams allows to model concrete points in time and durations and to express constraints on those timing characteristics. In addition, interaction diagrams also allows to model constraints for durations. Consequently, it is possible to express requirements for the timing behaviour of software systems in standard UML. Figure 3.3 for instance shows a sequence diagram with concrete points in time (TimeObservation) together with their constraints (TimeConstraint) as well as durations (DurationObservation) together with their constraints (DurationConstraint).

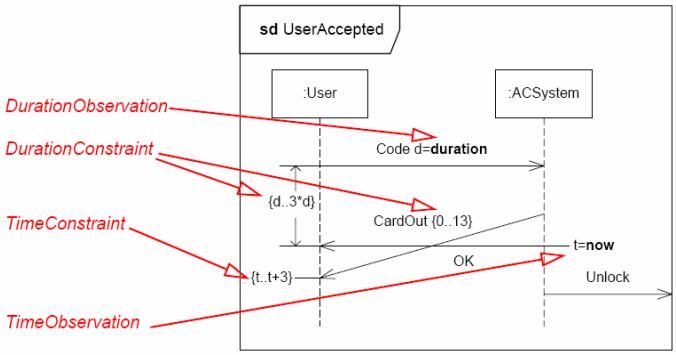

Figure 3.3: Example of a Sequence Diagram with Time Specifications

Thus, one important quality characteristic can be expressed in standard UML. Other characteristics such as reliability and availability are added to the UML via specific profiles described below.

3.2.1 UML Profile for Reliability

Cortellessa and Pompei propose a UML profile for Reliability [33] that was later integrated in a UML profile for quality of services (see Section 3.2.3). The authors' indention in developing this profile was to support software developers using commercial of the shelf components. The profile provides means to predict the reliability of the entire system based on the individual reliabilities of the system's components.

Description of the Model

Cortellessa and Pompei define a conceptual model of reliability clarifying how reliability can be described (see Figure 3.4) and show how this conceptual model can be implemented in UML.

The conceptual model consists of the classes `REservice`, `REuser`, `REcomponent`, `REhost` and `REconnector`. The prefix `RE` stands for reliability. In the view of the authors each `REservice` consists of `REcomoponents`, which interact with each other with the help of `REconnectors`. A `REconnector` is a logical connection between two `REcomponents`. Each `REcomponent` can then be decomposed hierarchically and is executed at one `REhost`. Each `REhost` however can run more

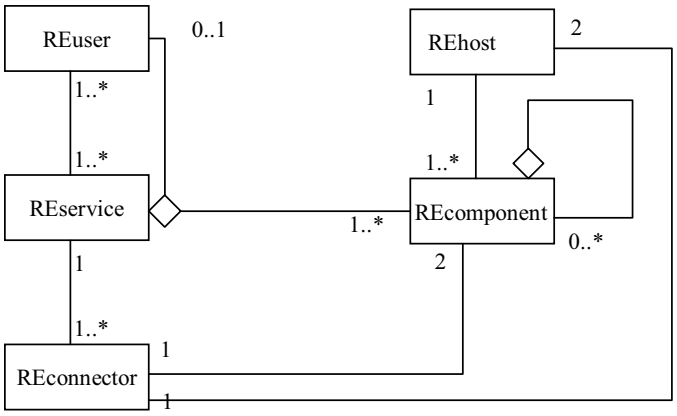

Figure 3.4: Proposed Conceptual Model for Reliability [33, p. 199]

than one `REcomponent`. In addition to its logical connection, a `REconnector` can also be used to model the physical connection between two `REhosts`. A `REservice` defines an interaction of one or more `REcomponents` and is always triggered by a `REuser`.

The relation of this conceptual model to the UML is twofold: Firstly the classes of the conceptual model are derived by sub-classing from existing UML classes [33, p. 200]. In addition the conceptual model is extended by stereotypes. The tagged values associated with these stereotypes represent the quality attributes associated with each stereotype. The tagged values are listed in Table 3.4.

Context	Quality Attribute	Definition
REComponent	REcompfailprob	Atomic failure probability of the component.
	REbp	Number of invocations on the component (busy periods).
REConnector	REconfailprob	Atomic failure probability of the connector.
	REnummsg	Number of invocations of the connector (number of messages).
REUser	REaccessprob	Probability that a user of a certain type accesses the system.
	REserviceprob	Probability that a user uses a certain service.
REService	REprob	Probability that a service is required.
REhost	REindexHost	Names of the hosts to which the host describe is physically connected.

Table 3.4: Tagged Values of the Reliability Concepts

Relevance for S-Cube

By defining the above-mentioned tagged values it is then possible to calculate the probability of the failure of a service $FP(S)$ by applying the following formula:

$$FP(S) = \sum j = 1..K(\prod i = 1..N(1 - REcompfailprob(i))^{REbp(i,j)} * \prod (\forall(l,i))(1 - REconnfailprob(l,i))^{REnummsg(l,i,j)}).$$

Although the UML extension proposed by Cortellessa and Pompei provides a rigorous way to calculate the failure probability of the entire software system, it is limited to those quality characteristics regarding the reliability of a software system. In addition, it mostly concentrates on probabilities, which can easily be used for calculations. Other reliability parameters, however, are not addressed (see also Table 3.5).

Goal	Relevance	Rationale/Explanation
Quality Definition	o	The main focus of this quality model is to provide means for describing the reliability of software systems.
Quality Negotiation	-	There are no techniques to negotiate quality.
Quality Assurance	o	The analyses methods provided allows to calculate the failure probability of a single service.
Discipline(s)	Software Engineering	

Table 3.5: Potential Relevance of the Reliability Profile for the Goals of WP-JRA-1.3 (+ = fully relevant, o = partially relevant, - = not relevant)

3.2.2 UML Profile for Schedulability, Performance, and Time Specification (SPT Profile)

The UML Profile for Schedulability, Performance, and Time Specification (SPT-profile) [100] produced by the OMG is an extension of the UML to cover performance aspects. It resembles various proposals to model performance in the UML, which were put forward e.g. by Williams and Smith 2002 in [130], Dimitrov et al. in [40], Bertolino et al. [15] and Gu and Petriu in [56]. It was designed to model real time characteristics of software systems in UML in a precise way to enable analysing the performance of the software system prior to its implementation.

The three roles defined in the model are modellers, model analysis method providers and infrastructure providers. Each of these roles is described below:

1. Modeller: The main focus is on the modeller who uses the SPT profile to model performance characteristics of a software system and uses these characteristics to predict if the system satisfies the performance requirements.

2. Model analysis method provider: The model analysis method providers are the teams who provide concrete techniques and/or tools to predict the performance of the software system described by the SPT profile.

3. Infrastructure Provider: The infrastructure provider are those persons who provide run-time technologies such as Real-Time CORBA to which the performance characteristics in the SPT profile should be related for precise performance predictions.

The SPT-profile is described in nine sections. Two of the sections relate to quality attributes of interest to S-Cube: in Section 4 time modelling is described and Section 7 describes performance modelling. The modelling of time was described in Section 3.2, the remainder of this section concentrates on performance modelling.

Description of the Model

An example of an SPT annotated UML model is given in Figure 3.5. The example depicts for instance that the response time of the first activity of the `Browser` object, i. e. the time between resource request and resource release, should be less than 500ms in 95 % of the cases. In addition, the first step of the `VideoServer` object has an estimated demand on its host resource of 1ms. In a similar fassion the first activity of the `VideoPlayer` object has an estimated average deoman on its host reosurce of 15ms with an allowed standard deviation of this value of at most 10ms.

The SPT profile provides means to [100, p. 7-1]:

1. describe performance requirements in software designs,

2. associate quality attributes (called quality characteristics in the profile definition) with UML elements,

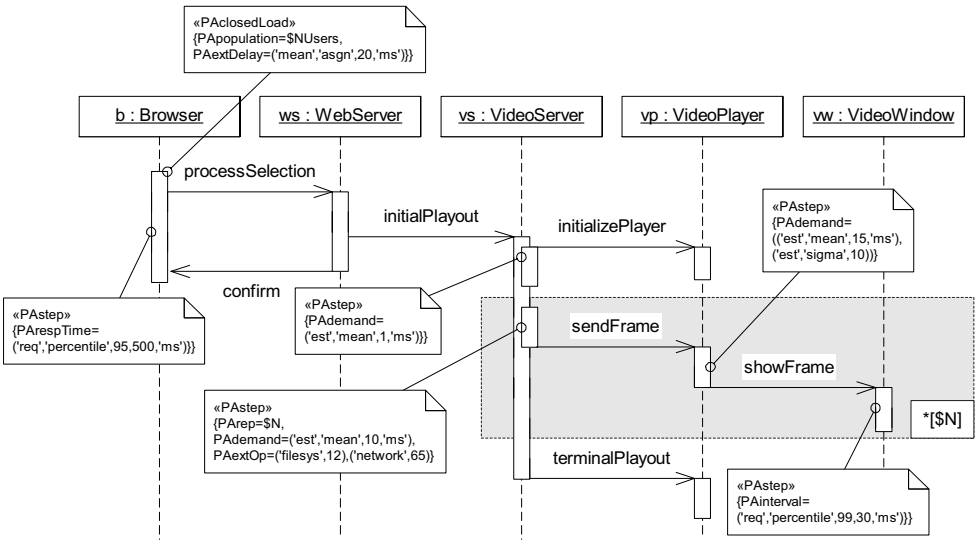

Figure 3.5: Example of a SPT annotated UML Model [100, p. 7–28]

3. specify execution parameters used to compute and predict the performance of the software system and

4. represent predicted performance results in modelling tools.

Figure 3.6 presents the conceptual model, which defines the SPT's concepts to describe performance attributes of software systems (labelled performance analysis domain model in [100]). The central element in the model is the `PerformanceContext`. It specifies some scenarios that define the context in which the performance of the software system should be evaluated. Each scenario is defined as a sequence of steps (`PStep`) representing a single activity of the system.

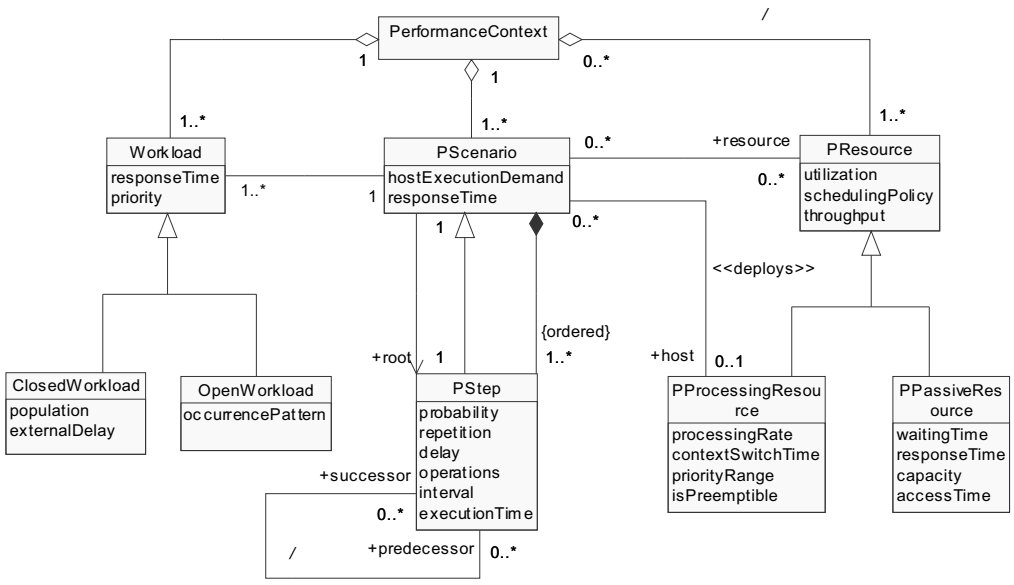

Figure 3.6: Performance Analysis Domain Model [100, p. 7-5]

Each scenario uses resources (`PResource`) and is executed under a certain workload (`Workload`). The resources are distinguished in passive and active resources (`PPassiveResource` and `PProcessingResource`). The workload is further distinguished into open and closed workloads (`OpenWorkload` and `ClosedWorkload`). The distinction between open and closed workloads is that open workloads are based on a probability distribution of system usage while closed workloads describe a concrete system usage (e.g., 10 users use the system each 10 minutes for 1 minute).

Quality attributes are defined as attributes of the conceptual model's classes, which are subsequently mapped to tagged values of the classes' stereotypes, which make up the SPT-Profile. The attributes are explained in Table 3.6.

Context	Quality Attribute	Description/Definition
Scenario	hostExecutionDemand	execution demand of the scenario on its hosts in case that every step in the scenario is executed on the same host.
	responseTime	time to execute the scenario
Step	hostExecutionDemand	execution demand of the step on its host resource
	delay	delay of the step, e.g.due to a user interaction
	responseTime	delay to execute this step
	probability	probability to execute this step in case of a fork in the workflow
	interval	time between the execution of this step
	repetition	number of times the step is repeated
	operations	used to express operations on the resource, which is not explicitly modelled
Resource	utilization	computed amount of resource utilisation calculated based on the entire model
	throughput	rate at which the resource performs its function
	schedulingPriority	policy to control access to resource
ProcessingResource	schedulingPolicy	see resource. pre-defined policies are: FIFO, HeadOfLine, PreemtResource, ProcSharing, PrioProcSharing, LIFO
	processingRate	speed factor for the processor
	contextSwitchTime	time required to switch from one scenario to another
	priorityRange	set of priorities for the processor
	isPreemptable	shows whether the processor is preemtable
PassiveResource	schedulingPriority	see resource. pre-defined policies are: FIFO, PriorityInheritance, NoPreemption, HighestLockers, PriorityCeiling
	capacity	number of concurrent users
	accessTime	time to acquire and releasing resource
	utilization	mean number of concurrent users
	responseTime	time between the resource request and resource release
	waitingTime	time between access request and granted access
Workload	responseTime	time between start and end of a scenario
	priority	priority of the workload
OpenWorkload	occurrencePattern	pattern to describe the interarrivel times between consecutive instances
ClosedWorkload	population	number of system users
	externalDelay	time between the response and the subsequent request

Table 3.6: List Quality Attributes in SPT [100, p. 7-7](Context = attribute's class)

Relevance for S-Cube

The SPT profile is relevant to S-Cube as it defines key quality attributes of software systems. SPT models can be used to analyse performance characteristics and therefore used as means of quality assurance (see also Table 3.7 on page 19).

Goal	Relevance	Rationale/Explanation
Quality Definition	+	The main focus of this quality model is to provide means for defining quality attributes for software systems.
Quality Negotiation	-	There are no techniques to negotiate quality.
Quality Assurance	+	The analyses methods enabled by the SPT profile allows to predict the performance of a software system and, thereby, assure its performance quality attribute.
Discipline(s)	Software Engineering	

Table 3.7: Potential Relevance of the SPT Profile for the Goals of WP-JRA-1.3 (+ = fully relevant, o = partially relevant, - = not relevant)

3.2.3 UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QoS Profile)

Many proposals were put forward to model quality attributes with the Unified Modelling Language in response to the Object Management Group's (OMG) request for proposals for developing a UML quality of service (QoS) profile to specify Service and Fault Tolerance Characteristics and Mechanisms (UML QoS profile) [2, 35, 7]. Especially the work described in [7] was mainly adopted in the UML QoS Profile issued by the OMG in 2006 [102]. The QoS profile is primarily an approach to describe how quality should be defined. However, the profile comes with proposals for concrete quality attributes on which we focus here.

The UML QoS profile provides a unified framework for specifying quality attributes in UML models. Annotating UML models with quality attributes is described as a three step process [14, 43]:

1. The UML QoS Profile provides a catalogue of quality characteristics. This catalogue must be tailored to the project at hand.
2. The tailored catalogue of quality characteristics is then used to derive a project-specific quality model.
3. This project-specific quality model is in turn used to annotate UML models.

Description of the Model

The document describing the UML QoS profile is organised in 14 sections. The most relevant sections for our purposes are: Section 8 containing a coarse-grained description of the QoS quality modelling language and Section 10, which contains the catalogue of QoS characteristics deemed relevant by the OMG.

The QoS Characteristics model depicted in Figure 3.7 distinguishes between quality characteristics (`QoSCharacteristic`), quality dimensions (`QoSDimension`) and quality parameters (`QoSparameter`). Quality characteristics are used to represent quantifiable properties of services, which "...are specified independently of the elements that they qualify." [102, p. 10] Consequently, quality characteristics are defined independently of any UML model and will be associated with these UML models via annotations. `QoSCharacteristics` are used to describe general non-functional properties (e.g., latency), which can be specialised and tailored for a concrete project.

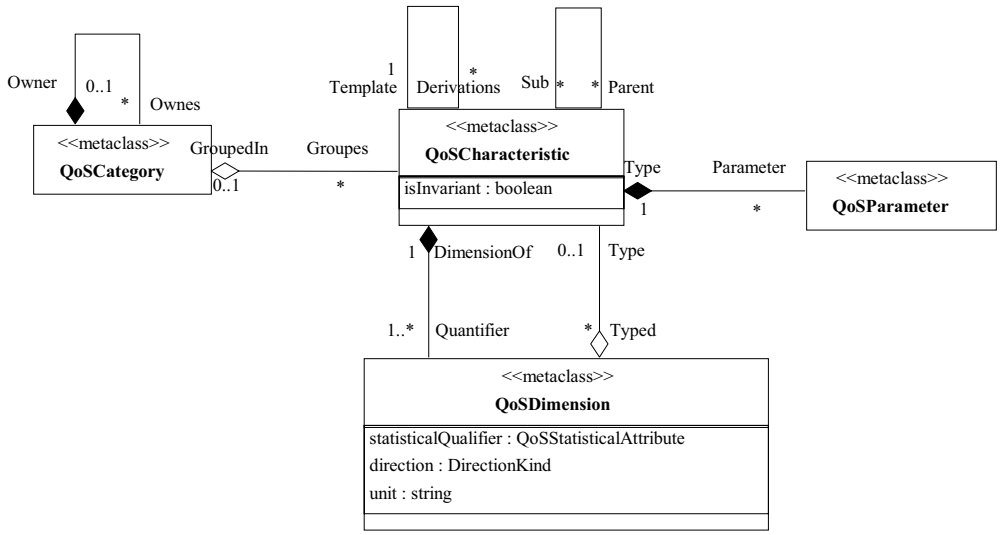

Figure 3.7: QoSCharacteristics Model [102, p. 11]

While `QoSCharacteristics` describe *what* should be measured, `QoSDimensions` describe *how* a single criterion should be measured. `QoSDimensions`, therefore, specialise `QoSCharacteristics`. The quality characteristic reliability for instance may be refined by the dimensions time to repair and time to failure e.g., [74, 47, 46].

The general quality attributes provided by the OMG in Section 10 of the document are listed in Table 3.8 on page 21 [102, p. 31]. Because of the general nature of the QoS profile the quality attributes do not depend on a particular context. Consequently, the context column is omitted here.

Quality Attribute	Definition
Performance	Performance describes the timeliness aspects of the software systems behaviour. It, therefore, includes quality attributes such as latency, throughput and turn-around. Performance may also be used to describe the utilisation of resources in order to provide a particular service (e.g.memory and CPU usage). The latter performance aspects are refined by efficiency and demand.
Throughput	Throughput refers to the number of event responses handled during an interval. Througput can be further distinguished into input-data-throughput (arrivalal rate of user data in the input channel), communication throughput (user data output to a channel) and processing throughput (amount of data processed).
Latency	Latency refers to a time interval during which a response to an event must arrive.
Turn-around	The turn-around specifies the absolute time limit required in fulfilling a job, task or service (e.g.the worst case scenario).
Efficiency	The efficiency is the capability of the software to produce their results with the minimum resource consumption.
Demand	Demand specifies how much a resource or a service is needed.
Dependability	Dependability covers all properties of a software system to assure that it delivers a certain service reliably. Conseuqently, the dependability quality attribute covers other quality attributes such as availability, reliability, fault-tolerance, recoverability and maturity.
Availability	Availability describes the ability of a software system to constantly provide certain services. It can be further refined by the mean time between failure and the time needed to recover from a fault. Availability further subdivided into connection-availability and processing availability.
Reliability	The capability of the software product to maintain a specified level of performance when used under specified conditions.

Fault-tolerance	A fault tolreant software systems contains mechanisms to detect and correct latent errors before they become effective.
Recoverability	Recoverability of a software product is the ability to recover affected data in caise of a failure.
Maturity	Maturity is the capability of a software system to avoid faulres as a result of faults.
Security	Security covers the capability of a software system to protect entities (e.g.data) and to protect the access to resources (e.g.printers). Security can be refined to access-control and protection.
Access control	Access-control specifies the control policy used for the access to services (e.g.security levels).
Protection	Protection describes more generally the methods used to grant access to a service and the probability of a control break.
Accuracy	Accuracy describes the maximum allowed distance between the expected and the delivered results.
Coherence	Coherence includes characteristics about concurrent and temporal consistency of data and software elements.

Table 3.8: General Quality Attributes Defined by the OMG [102, p. 31]

Proposed Structure of the Quality Attributes

The quality attributes are organised in the three packages `dependability`, `performance` and `functionality` (see Figure 3.8).

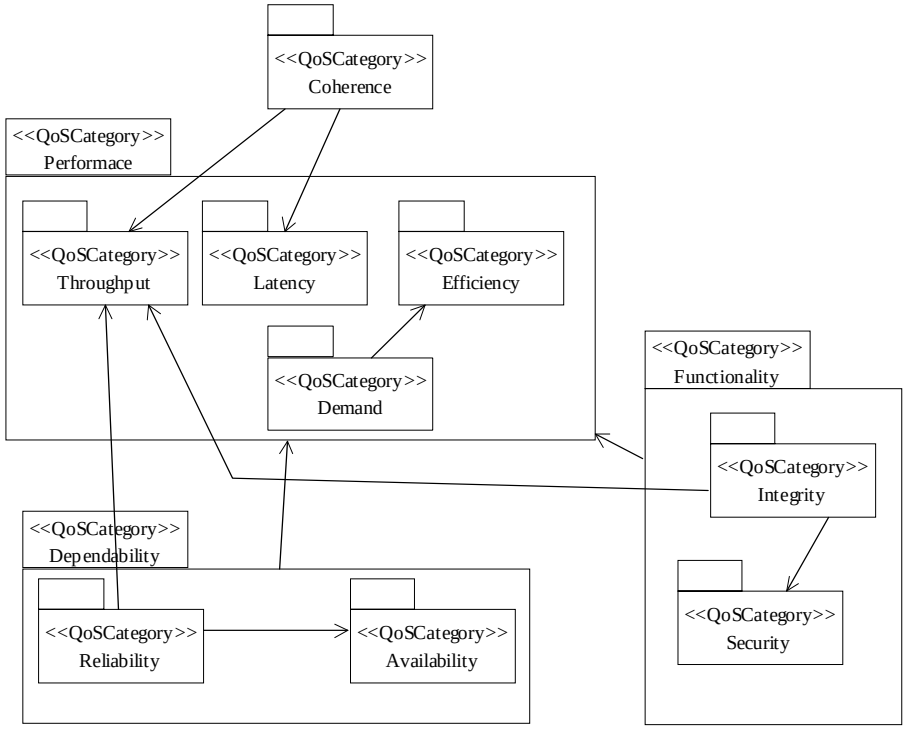

Figure 3.8: Structure of the Quality Attributes [102, p. 32]

Relevance for S-Cube

Since many organisations contributed to the OMG profile it is expected that the quality profile is quickly adopted by industry and academia and, as a consequence, it becomes relevant for the S-Cube project (see also Table 3.9).

Goal	Relevance	Rationale/Explanation
Quality Definition	+	The main focus of this quality model is to provide a language for specifying quality attributes for software systems.
Quality Negotiation	-	There are no techniques to negotiate quality.
Quality Assurance	-	There are no specific techniques to assure the quality of software systems.
Discipline(s)	Software Engineering	

Table 3.9: Potential Relevance of the QoS Profile for the Goals of WP-JRA-1.3 (+ = fully relevant, o = partially relevant, - = not relevant)

3.3 Statically Inferred QoS Attributes (Software Engineering / SOC)

In this section we will focus on the quality aspects of a service whose attributes (and other related properties) are amenable to being statically inferred [96].

While non-trivial and interesting software properties are known not to be statically decidable, it has been the case that many useful, safe approximations of relevant properties can be automatically inferred ahead of run-time (i.e. without dynamically executing the software component), from sufficient information on that component, and the components it interacts with. The process of inferring these properties is known as static analysis. The same principle applies to Web services as specific kinds of software components.

Static analysis is performed by an analyser that takes the source code of a Web service, along with other information, performs analysis, and infers its QoS attributes and other properties.

Description of the Model

Static inference of QoS attributes for Web services can be performed at *compile time*, before services and related resources are packaged for distribution. However, in complex and dynamic service environments, some forms of analysis can be also performed at *deployment time*, in cases where distributions essentially contain all required information (including source code); an example would be a distribution of service assemblies containing BPEL process definitions. For compiled services, written in languages like Java, C#, etc., the service infrastructure normally does not have access to the source code, and therefore, to infer, prove or check properties of the service, it needs access to additional information, and it may even decompile and analyse service byte-code [4, 3].

Another valuable kind of information needed for static analysis of a service relates to services it interacts with. While this is not strictly necessary, it would of course improve the accuracy of the analysis results, as more information regarding how the service under scrutiny interacts with its environment will be available.

One obvious option, assuming a suitable analysis for the relevant properties and the programming language at hand is available, is to simultaneously analyse these interacting services [81, 80, 32, 112]. This makes it necessary for the code defining these services to be available to the analyser. However, the circumstances under which such simultaneous analysis can happen are usually restricted: at compile / design time, a developer would normally need access to the source code of these services, and at deployment time the services would need to either belong to the same service assembly, or be deployed on the same server.

The hairiest issue issue in real service execution environments is probably that it may not be possible to guarantee which particular service implementation, conforming to the same functional interface, will be selected for pairing. In principle, such guarantee cannot be given in the generally decentralized and distributed dynamic environment in which services operate.

One way to resolve the lack of information needed for the inference or proof of service properties is to require that services are deployed together with a set of *assertions* that express properties of the service and which should have been previously generated at compile / deployment time.

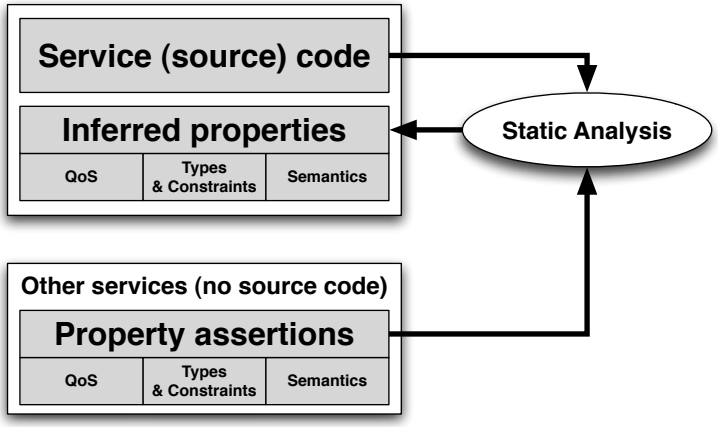

Figure 3.9: A schematic view on static inference of service properties

Among the properties which impact quality of service (in a broad sense), the following (non-exclusively, of course) may be inferred by using some kind of analysis tools:

Resource consumption: Various QoS attributes of a service can be expressed as properties which can be statically analysed. In particular, particular those which are related with generalized resource consumption [91, 92] can be used to predict, within a certain accuracy, the behaviour of a service or service composition. Those whose consumptions grows monotonically can be seen as particular instances of computational complexity, and include, for example, number of messages sent or received, the total size of transmitted message, the total time waiting for an answer to arrive, etc.

Types and Constraints: Typing of in-bound and out-bound messages for services is usually based on a set of structural constraints contained in an XML Schema, which are imposed upon XML messages that are exchanged [65, 64].[1] Structural constraints may not be able to express all required relationships between nodes (elements and attributes) in an XML document.[2]

A richer typing approach would allow structural typing information to be complemented with relationships between the message contents [22]. By making this additional information an integral part of service description, available at assembly, deployment and run-time, outgoing and ingoing messages can be inspected to ensure that they have been created as required (for example, by means of runtime checks) and not only from a structural point of view. This would thereby increase robustness and overall quality.

More expressive type systems, which carry with them some of the intended semantics of the data, can also be used when assembling services (either at design time or dynamically, as e.g., when an adaptation is needed) in order to select those which are more adequate for the purpose at hand. This has a clear relationship with the objectives of WP-JRA-2.2.

[1] XML Schemata also allow some referential constraints to be stated, by means of ID and IDREF attribute characterization.
[2] Or, in general, represented with any other syntax.

Other Semantic-Related Properties: This includes service properties which do not fall under the above two categories and which can be used to characterize QoS of a service or service composition, while being amenable to inference by automatic static analysis.

Proposed Structure of the Quality Attributes

The usability of static inference of QoS attributes for Web services can be additionally enhanced in the context of end-to-end QoS provision by means of:

Inference of safe bounds approximations: Static inference of QoS service attributes in the form of safe lower and upper bound approximations, can produce guarantees that the actual QoS attribute will behave within these bounds approximations. These guarantees can then be used for deducing safe approximate grounds of QoS attributes for composite (or aggregated) service.

Input data awareness: Calculation of QoS attributes can be enhanced by taking into account features (dimensions or metrics) of messages received by the service, that is, the message which initiated service execution, as well as other potential messages accepted from the initiator (service client) in the course of service execution.

Input data awareness leads to a significantly more refined and finer grained insight into service QoS attributes and behaviour, and to descriptions of the safe bounds approximations for QoS attributes in terms of functions over the input data.

Parametricity: Statically inferred QoS service attributes can be further improved by inclusion of parameters that cannot be statically inferred, but depend dynamically on features of the service execution environment (network speed, latencies, throughput, degree of parallelism, etc.).

Inclusion of such run-time parameters in QoS attribute descriptions allows service infrastructure to make better predictions and matchmaking decisions, at the cost of supplying actual values of parameters at runtime, based on monitoring.

Relevance for S-Cube

Static inference of QoS attributes for Web services relates to several key research components of S-Cube. First, it brings techniques and approaches of static analysis to the field of end-to-end QoS provision. Second, static inference of QoS attributes as functions of input data can be useful for inference of the key performance indicators (KPIs), within the S-Cube component dealing with BPM. Third, it can be employed for inferring and using safe QoS bounds functions on input data for service compositions. And, fourth, by integrating the inferred QoS functions with run-time parameters, static inference can lead to a more precise, predictive monitoring.

3.4 Design by Contracts Models (Software Engineering / SOC)

Contract-based approaches have been used to enforce general and quality related constraints in object oriented architectures and component-based systems. We envision its applicability to enforce quality related constraints in service oriented architectures. In this section we review work on application of contracts to object-oriented and component-based architectures with special emphasis on quality characteristics.

A contract is an assertion on a combination of properties in a program. An assertion should always be true if we wish to maintain the correctness of the program execution. The notion of a contract originates in work [44] and Hoare [60].

Goal	Relevance	Rationale/Explanation
Quality Definition	+	The model statically inferred QoS attributes defines QoS attributes in terms of safe upper and lower bounds (potentially parametrized with attributes of the run-time system) of generalized resource consumption, as functions of input data.
Quality Negotiation	+	QoS expressed as functions of input data allow more expressive and flexible negotiation rules and protocols, taking into account the profile of the actual processing tasks.
Quality Assurance	+	Functions expressing safe upper and lower bounds of QoS attributes allow QoS analysis for service composition and can be used to detect possible violations in advance
Discipline(s)	Software Engineering, SOC	

Table 3.10: Potential Relevance of Static Inference of QoS Attributes for the Goals of WP-JRA-1.3 (+ = fully relevant, o = partially relevant, - = not relevant)

In object-oriented software, assertions play an important role in the proper definition of types. Preconditions, postconditions and class invariants help to define operation types in class interfaces. The Eiffel language developed by Betrand Meyer's [87] contains native constructs to deal with the description of preconditions, postconditions, class invariants, loop invariants and other assertions. These constructs define a sound design process namely *design by contract* by B. Meyer. More recent approaches use OCL for UML [55] or JML for Java [78].

Description of the Model

In this section we first present an overview of contracts and then focus on the application of contracts to specify invariants, pre-conditions and post-conditions for quality modelling in object-oriented and component-based software systems.

When contracts are applied to components, Beugnard et. al.[16] show how these contracts can be classified into categories that describe four levels of increasingly negotiable properties:

- The first level of contract is named *basic or syntactic level*. Basic contracts address the common type specification. Interface definition languages (IDLs) and typed object-based or object-oriented languages let the component designer specify the operations that a component can perform, the input and output parameters each component requires and the possible exceptions that might be raised during operation.
- The second level is named *behavioural level*. Behavioural contracts improve the level of confidence in a sequential context. Drawing from abstract data type theory, designers can specify an operation's behaviour using boolean assertions, called pre- and postconditions, for each service offered, as well as for class invariants.
- The third level is named *synchronisation level*. Synchronisation contracts improve confidence in distributed or concurrency contexts. They specify global behaviours of objects as synchronizations between method calls. Synchronisation contracts aim at describing dependencies between services provided by a component, such as sequence, parallelism, or shuffle.
- The fourth level is named quality of *service level*. Quality of service contracts quantify quality of service and are usually negotiable. QoS contracts for provided properties and service depend often from QoS properties of the services required from the environment.

Historically, contracts were first applied to object oriented design. Design by contract includes several aspects:

1. Verification of the satisfaction of preconditions and postconditions of class operations;
2. Definitions of compatibility of assertions with subtyping.

Regarding meta modelling of contracts, various approaches have been proposed ranging from the reification of preconditions and postconditions without any management of subtyping or contract life cycle, up to contracts metamodels that include provisions for coordination specification and management. In this kind of metamodel, coordination invariants deal with the collective behaviour of the participants.

Modelling QoS of service using Level 4 contracts is the focus of this section. We envision the extension of the ideas presented in this section to model QoS in service-oriented architecture based systems.

Quality Attributes in Contracts

Syntactic level:

- *Reusability* The syntactic level relies on proper type definition, and type compatibility rules. Reusability is ensured by defining operation signatures that are general enough.
- *Trust* Types can be checked at compile time; therefore invocation of an operation will succeed if the system passed the compilation phase (although the method invocation of the operation may fail, for instance because of a level 2 contract violation). Invariants are used to model the type checking rules that ensure trust in the component; pre-conditions and post-conditions are used to ensure trust at interfaces of components in a composition, by modelling type checking rules.

Just as components are, services can be specified using invariants, pre and post-conditions to ensure their reusability and trust.

Behavioural aspect:

In this topic of binding the concept of design by contract and the concept of Components, Bertrand Meyer has proposed the concept of trusted component in [88]. He identifies two challenges to the research community:

- *Reusability* Pre-conditions and post-conditions are vital means of reusability support because they define a precise perimeter where existing components are useful. As with type definitions, the designers must try and define pre-conditions that are as weak as possible and post-conditions that are as strong as possible, in order to be able to quantify the quality of existing components.
- *Trust* Trust relies on pre and post-conditions as well but aims to produce components with fully proved correctness properties. Static checking of component service invocation is possible in some cases, but in the general case, testing is the only means available and it uses heavily these behavioural aspects of the contract system.

Use of contracts may offer additional benefits in the future, in the domain of services, where pre- and postconditions may form the basis of systems that allow dynamic negotiation of contracts among services.

Component synchronization:

Software components are processes that are executed at the same time and may exchange information. They are often distributed over several computers, and do not share a global clock. Each component is free to run at its own rate, but it must synchronize with other components when it needs to communicate. This is known as asynchronous parallelism [28]. Asynchronous parallelism has consequences for component reuse. A system built as an assembly of subsystems that have been proved correct is not itself necessarily correct. For example, the interaction between two components that do not have any internal deadlock may deadlock in combination if each awaits a message from the other.

Dealing with these problems of asynchronous composition is a major obstacle to component reuse. It is therefore necessary to specify component interfaces precisely using level-3 contracts, in order to capture the information necessary to detect deadlocks or livelocks, and then to prove the correctness of the assembly.

This problem is well-known and is not unique to the design of component based systems. It has been addressed in the telecommunications domain for the specification of new protocols and in the hardware systems domain for the design of multiprocessor architectures (e.g. cache consistency). Consequently many formalisms have been proposed for studying it, from theoretical models like Petri-nets [111] and I/O Automata [57, 83] to process algebras (CCS [90], CSP [61], LOTOS [69] or FSP [85]), or dedicated languages (SDL [71]) in the Telecommunications domain, or SystemC [121] for the design of hardware.

- *Reusability* Synchronization contracts express coordination requirements for interaction between a component and its environment. If these synchronization rules are not carefully designed, they can lead to rigidity of use of the component and therefore they can limit the reusability of a component.
- *Trust* Whenever a component implementation relies on a specific order of operation calls to perform a service correctly, the designer must provide synchronisation contracts that ensure that these operations are called in the proper order. Designer of component assemblies can then check that the composition of components obeys the various synchronisation requirements and that the assembly obeys the general service specification. This possibility provides a good support to the trust quality criterion.
- *Predictability* By composing synchronisation contracts, designers can check for deadlock freedom. This possibility improves the predictability of component assemblies.

In services domain, the problem of synchronization is still present and may require the use, as in component-based systems, of level 3 contracts definitions to avoid deadlocks in services compositions.

QoS specification of components assembly:

At the interface level of components, other properties more quantitative can characterize a component and its behaviour within an assembly. For example, if we consider a component that provides multimedia content, user satisfaction can be affected by latency, video rates or synchronization between sound and image. These extra-functional features are generally associated with QoS. Much work has been done on the specification of QoS, and other extra functional properties, within the component interface, such as SMIL (*Synchronized Multimedia Integration Language*) [63], TINA Object Definition Language [86] an extension of CORBA IDL for the QoS specification, or the well known QML (*Quality of service Modeling Language*) [48].

- *Reusability* Quantitative statements in component contracts typically provide means to choose between implementations (e.g. using the memory footprint required by a given component implementations). Typically, this helps designers to choose an adequately sized implementation. This possibility promotes reuse of components by easing the selection process for the component users.
- *Trust* In the component world trust is often related to timeliness of service execution, e.g. that all executions of a component operation will terminate after a delay specified during contract negotiation.
- *Predictability* Predictability has some correlation with trust but it may deals with stochastic properties, e.g. distribution of an operation response time. Predictability if often associated to performance evaluation, while trust is associated with safety.

All these works on QoS specification about components assemblies can represent a great input for services compositions quality definition.

Context	Quality Attribute	Description/Definition
Basic level	Reusability	Proper type definition, and type compatibility rules.
	Trust	Ensure trust in component and its interfaces using invariants, pre and post-conditions for type checking.
Behaviour level	Reusability	Precise definition of the scope of reusability of a component.
	Trust	Fully prove correctness of component properties
Component synchronization level	Reusability	Expression of the coordination requirements for interaction between a component and its environment
	Trust	Contracts ensure that operations on a component are called in the proper order.
	Predictability	Check for deadlock freedom by composing synchronisation contracts.
Service level	Reusability	Quantitative statements in component contracts providing means to choose between implementations.
	Trust	Often related to timeliness of service execution: termination delay specified during contract negotiation for example.
	Predictability	Often associated to performance evaluation.

Table 3.11: List Quality Attributes in Design by Contracts

Relevance for S-Cube

Contracts play a vital role in component-based architectures , as a key concept to describe components and their assemblies. In a service oriented perspective, contracts also play a key role at the business modelling level. Independently from the service implementation (which will most probably be component-based), designers need concepts to specify collaborations and quality of service. Service metamodels usually emphasize extensive means to define roles and responsibilities as well as contract life cycle; they favour dynamic behaviour against verification ((see also Table 3.12)

Goal	Relevance	Rationale/Explanation
Quality Definition	+	Quality is expressed in the form of contracts such as pre-condition, invariant, or a post-condition in a operation involving different components. The contracts are often expressed in a formal constraint language such as OCL [105]
Quality Negotiation	-	Contracts are used to specifying quality related constraints at component interfaces but they cannot perform negotiation of quality between components.
Quality Assurance	+	Quality can be assured by testing a component using test cases that satisfy pre-condition contracts. In [119] [17] the authors present pre-condition satisfaction as a means to automatically synthesize inputs to test a component. Automatic test generation for components can be extended to service oriented architectures in order to deal with unreliable service environments.
Discipline(s)		

Table 3.12: Potential Relevance of Design by Contract Models for the Goals of WP-JRA-1.3 (+ = fully relevant, o = partially relevant, - = not relevant)

3.5 Functional Quality in Service Composition (SOC)

The Service-Oriented Architecture, together with Web Service technology, concentrate on the definition of a software system as a complex distributed application that involves a composition of services. Composition rules describe how to compose coherent global services. In particular, they specify the order, in which, and the conditions under which, services may be invoked. The complexity of the composition model, its structure, and interactions between the involved services introduce composition-specific aspects to the application quality model. Moreover, the composite applications often span across the enterprise boundaries and involve various stakeholders. These stakeholders often come into play with their own requirements and policies that may be inconsistent or even contradictory in certain scenarios. The quality of the service-based system directly depends on the correctness with respect to the desired business requirements and expectations of all the interested parties. Figure 3.10 depicts the conceptual model of service composition. It captures each element that a service composition model should provide. A *composite service* must be described in terms of *Service Process* that is made of one *Activity*, a set of *Partners* that interact with the process and, *Data* used by the process. Activities represents the actions that a composite service executes to provide some functionality. Generalizing we can have a set of different activities that are used to accomplish different objectives in the composition. *Interaction Activity* is used when we want to invoke partner services or when we allow our partners to invoke our services. *Context Activity* permits to break up a process into separate units of work that represents execution contexts. Using a *Concurrent Activity* we can define a set of activities that will execute concurrently in the service process while *Conditional Activity* is used to choose one of the possible alternatives path within a process. The last concept is the *Error Activity*. It is the generalization of two different activities: *Fault Activity* and *Compensation Activity*. The first is used to enable the process to recover from abnormally terminated actions while the other attempts to undo the work of an activity or a set of activities that have already been successfully completed in response, for example, to a fault occurring.

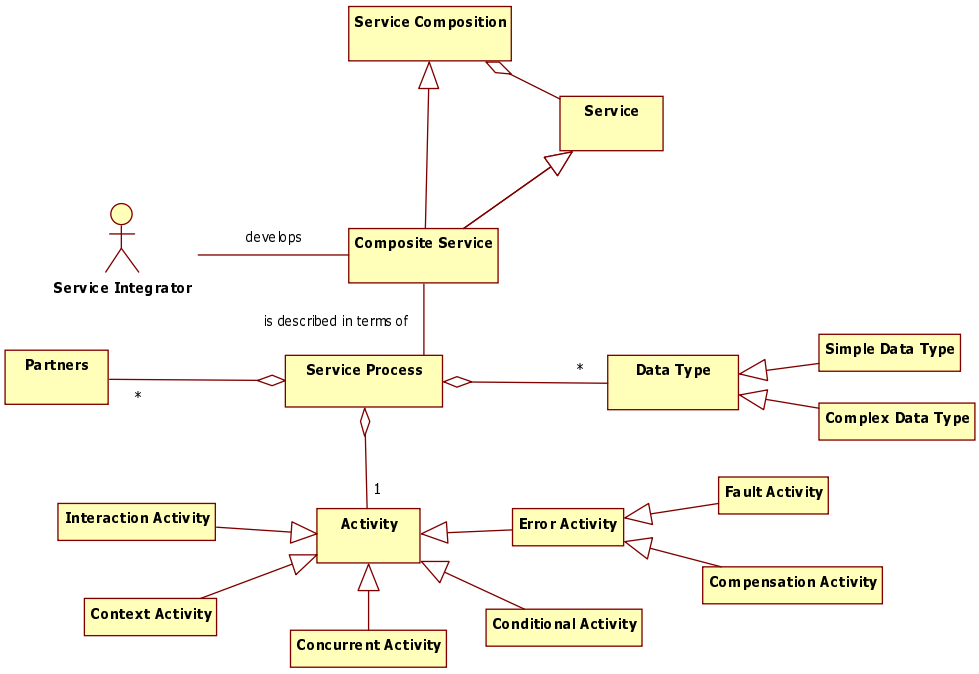

Figure 3.10: Service Composition Conceptual Model

Several organizations are developing languages for service composition, the most important one being the Web Services Business Process Execution Language WS-BPEL [18] (BPEL for short) and the Web Services Choreography Description Language WS-CDL [131]. Many of these languages however

have only a limited ability to support automatic service composition, mostly due to the absence of semantic representations of the available services. The Semantic Web community and others have proposed several solutions to these limitations, among which the Web Ontology Language for Web Services OWL-S [1] and the Web Service Modeling Ontology WSMO [132]. A problem of the previous languages is that they lack of software tools and approach to verify the correctness of service compositions. To solve this limitations, *formal methods* and tools can be used to decide whether services satisfy desirable properties and are interoperable. Moreover if one discover that a service composition does not match an abstract specification of what is desired, or that a main property is violated, this can be of help to correct a design or to diagnose bugs in a service. Recently several formal methods, most of them with a semantics based on transition systems (e.g. automata, Petri nets, process algebras), have been used to guarantee correct service compositions. A selective overview of the use of well-known languages and models by the formal methods community, to realize the approaches to service composition has been presented in [123].

Description of the Model

In this section we present principal characteristics that a composite service must provide to guarantee functional quality. As we show in Figure 3.11, to have quality in a service composition we must be able to verify that the composite service is *correct* and *interoperable*. Thinking about service composition we can have three different contexts in which we can define functional quality attributes: *workflow*, *dataflow* and *time*. This means that the satisfaction of three set of requirements must be accomplished. Moreover, for each contexts we have a set of functional requirements related to them that we explain in more detail below.

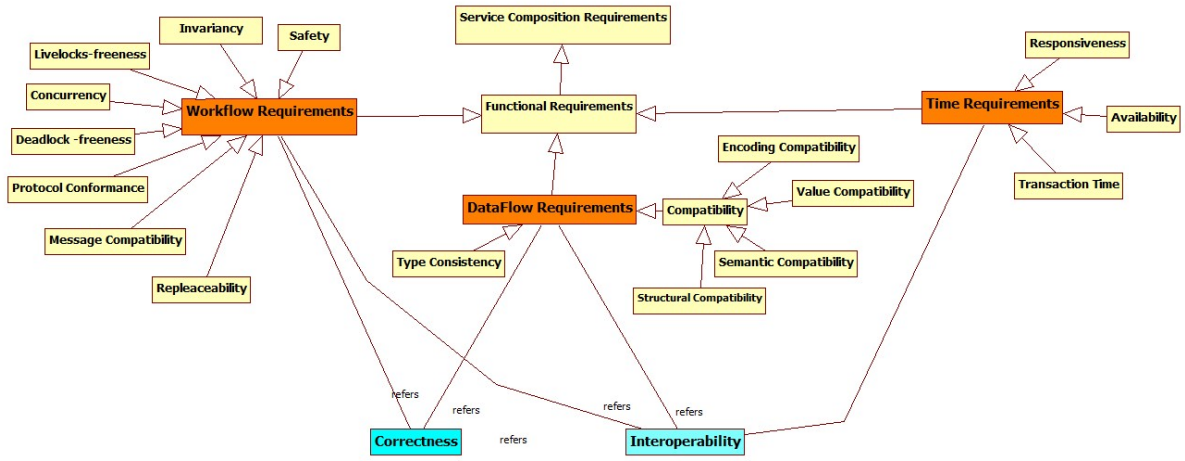

Figure 3.11: Functional Quality Concepts

Service composition may lead to large, complex systems of concurrently executing services. Functional properties of a new service can be determined and guaranteed, ensuring predictability and correctness of the resulting service properties.

Functional correctness for service compositions amounts to verifying the following aspects [89]:

- Type checking is ensured: Assure that the data flow in the composition is consistent with respect to definition of literal types.
- The composition invariants are preserved: The composed process should satisfy functional requirements;
- Composed process terminates correctly (e.g., absence of deadlocks or livelocks).

Composition requirements (choreography assumptions [113]) define the requirements, under which the component services participate in the composition. Functional requirements relate to the purpose of the composition, i.e., the task it has to accomplish. They restrict the valid behaviors of the composition component, and the actual executions of the composition should correspond to them.

Composition requirements may refer to different aspects of the composition execution:

- General requirements, which represent constraints and rules according to the *business logic* of an application.
- *Transactionality* requirements (e.g., define the consistency and/or atomicity rules).
- The *concurrency* and *coherence* requirements (e.g., the rules and constraints on message ordering and relations).
- The *data quality* properties (e.g., constraints on the data flow and data dependencies across the execution of a composition: integrity, validity).
- The *time quality* properties, such as constraints on activity, transaction, or process durations (latency, response time, transaction time).

According to the classification proposed in [36] and [123], the following types of correctness requirements may be considered:

- *Safety* properties express that something (bad) never occurs. These properties include assertions, invariants, properties that define a sequence of events, properties that check values of variables, and properties that deal with resource allocation.
- *Liveness* properties assert that some event does eventually happen.

Correctness of service composition requires that the properties of the composed service must be verified. In order to ensure functional correctness of the service composition, all the existing quality assurance techniques may be foreseen: testing, static analysis, and monitoring. Depending on the requirements type and addressed aspects, these techniques may use different formalisms and capabilities to represent and analyse the requirements.

The standardisation of Web Services makes their interoperability easier, however the problem of service compatibility remains. While the lower levels of the interaction stacks are standardized, different Web Services may still support different interfaces and protocols. In fact, having loosely-coupled and B2B interactions imply that services are not designed having interoperability with a particular client in mind. They are designed to be open and possibly without knowledge, at development time, about the type of clients that will access them.

Interoperability among Web Services, just like interoperability in any distributed system, requires that services use the same (or compatible) protocols, data formats, and semantics. With respect to service *data incompatibility*, the following types of mismatches may be identified [114]:

- *Structural* incompatibility is a mismatch in the structure of the (XML) message sent by the sender and expected by the receiver.
- *Value* incompatibility arises when the structure is as expected, but the filled-in values are unexpected.
- *Encoding* incompatibilities arise because instances belonging to different schema types are not identical even if they have the same structure and identical values.
- *Semantic* incompatibilities arise when different vendors introduce extensions with identical syntax (i.e., same structure and value) but differing meanings.

Service *interface incompatibilities* arise when the services provide operations that have the same functionality but differ in operation name, number, order or type of input/output parameters.

Service *behavior incompatibilities* arise when the services have different business protocols [12]. A business protocol specifies message exchange sequences that are supported by the service, for example expressed in terms of constraints on the order in which service operations should be invoked.

There exist several notions of quality characteristics related to the problem of interoperability:

- *Compatibility* amounts to the necessity of two interacting parties to fulfill each other's constraints and, therefore, to correctly interact.
- *Substitutivity* (replaceability) requires that one service demands fewer or fulfills more constraints than another service and therefore can substitute it in the composition.
- *Reliability* amounts to the ability to deliver responses continuously in time (service reliability) and the ability to correctly deliver messages between two endpoints (message reliability).
- *Accessibility:* The responsiveness towards service requests.
- *Exception handling/Compensations:* What happens in case of an error and how to undo the already completed activities. Since services are usually long-running processes that may take hours or weeks to complete, the ability to manage compensations of service invocations is critical.
- *Conformance* amounts to the necessity of the services participating to the composition to respect the global protocol. That is, all the behaviors generated by the services should conform to the conversations foreseen by the protocol model.

In order to ensure the interoperability requirements, both static and dynamic analysis means are foreseen. Static Analysis techniques (e.g., model checking) are used, for instance, to check the compatibility or substitutivity requirements. Dynamic analysis techniques (e.g., run-time monitoring) are used to check the conformance at run-time, in order to guarantee that the services do not violate the global protocol or their own published protocols (e.g., domain monitoring in [113]).

Table 3.13 summarizes all functional quality attributes introduced above categorized respect to the context in which are relevant. Its first column shows three contexts in which we can define various quality attributes that are listed and described in the other two columns. We can notice that each context and each attribute in this table have already been depicted in Figure 3.11 as concepts.

Relevance for S-Cube

S-Cube's goals include describing the functional quality of service compositions. The main problems with most practical approaches to service compositions are the verification of correctness and the analysis of interoperability. As we said in Section 3.5, due to their solid theoretical basis, the use of formal methods can increase the correctness and the interoperability of service compositions. Moreover, the tool support that comes with them allows the simulation and verification of the behaviour of a model at design time, thus enabling the detection and correction of errors as early as possible and before *before implementation*. Table 3.14 summarizes the relevances of these functional quality characteristics for S-Cube.

3.6 Service Networks and KPIs (BPM)

In this section we present Service Networks which is a model situated on the BPM level of the S-Cube Layering. Its quality is assessed in terms of Key Performance Indicators (KPIs) which measure quality attributes on the business level (a.k.a. quality of business (QoBiz); cf. Chapter 2).

Context	Quality Attribute	Description/Definition
WorkFlow	Deadlock-free	Never happen that none of the services in the composition meet the condition to move to another state and all the communication protocol are empty.
	Livelocks-free	Some event does eventually happen.
	Invariancy	The composed service should satisfy functional requirements.
	Safety	Assertions that some undesired event never happens in the course of the workflow.
	Protocol Conformance	Successfull conversation among services.
	Message Compatibility	No mismatch among messages sent and received.
	Repleaceability	The ability to substitute a service, in such a way that the change is transparent to external clients.
	Concurrency	The ability to perform concurrent processing.
DataFlow	Type Consistency	The data flow is consistent with respect to definition of data types.
	Compatibility	The necessity of interacting services to fulfill each other's constraints and, therefore, to correctly interact.
	Structural Compatibility	XML messages sent by the sender and expected by the receiver have the same structure.
	Value Compatibility	When two Web Services have the same structure and identical filled-in values.
	Encoding Compatibility	When Web Services instances, belonging to different schema types, are indentical and have the same structure and values.
	Semantic Compatibility	When different vendors introduce extensions with identical syntax (i.e., same structure and values) and the same meanings.
Time	Responsiveness	A service is able to response when invoked.
	Transaction Time	Represents the transactional properties of the service.
	Availability	The probability that a service is available at any given time.

Table 3.13: List of Functional Quality Attributes

In today's networked economy, there is an increasing need for companies to act together in a concerted manner to meet the needs of their joint customers. Thereby, interacting companies build networks, focusing on optimizing their financial benefits both at the individual and the network level. Service (Value) Networks (SNs) is an approach for modelling such networks and for analysing and optimizing company's business collaborations [26]. A business environment is modelled as a graph consisting of business partners and their relations depicting partners as nodes and their offering and revenues as edges. An SN offers services that are obtained by composing other services provided inside the SN by different partners.

Modelling a business landscape as SN allows both calculating the value gained by a single partner when joining the collaboration network and measuring the value of the whole network. An SN is not necessarily modelling interactions between different companies, but can also be used for modelling different units in a company. For example, a company could use the SN to model its internal business unit structure as SN, measure the profitability of the structure and perform some changes for improvement, e.g., decide to in-/outsource some units.

Description of the Model

Casewell et al. proposes an approach on how to calculate value of the SN by taking into consideration also customer satisfaction in addition to financial metrics [26]. The value of an SN is a KPI (Key Performance Indicators) that can be measured for the SN as a whole and for a single partner in the SN based on the following KPIs: revenues resulting from offering of certain services (which can be tangible and intangible), cost for providing these services, and customer satisfaction. These KPIs are again calculated based on a set of lower-level KPIs. KPIs can be regarded as quality attributes of an SN.

Goal	Relevance	Rationale/Explanation
Quality Definition	+	The main focus of functional quality model is to define quality properties that we want to verify to guarantee that a software systems is correct and that each piece of software is interoperable.
Quality Negotiation	o	To guarantee interoperability among services we can use existing automatic techniques [20, 13] to generate adapters capable of solving behavioral mismatches in a service composition.
Quality Assurance	+	The analysis methods enabled by formal methods allows to predict the correctness and interoperability of a software system and, thereby, assure its functional quality attribute.
Discipline(s)	Software Engineering	

Table 3.14: Functional Quality for the Goals of WP-JRA-1.3 (+ = fully relevant, o = partially relevant, - = not relevant)

In Table 3.15 we present a possible classification of KPIs and some typical example KPIs. This classification is used in the Balanced Scorecard [73], which is a popular performance measurement instrument in the business world. The KPIs are thereby grouped according to four dimensions: financial, customer-facing, operational, and specific to "learning & growth".

Context	Quality Attribute	Definition
Financial	Revenue	Income of the company received from the sale of goods or services during a certain period of time. Revenue can be expressed as a function of the price per unit of the good for sale and of the amount of goods that have been sold in a time period
	Cost	Amount of money that is spent to produce or provide goods or services during a period of time. It is a function of supplies, services, labour and equipment used in a time period
	Relationship Value	Value that is expected to be acquired due to all companies' relationships and interactions with other economic entities within the value network during a period of time ([26]). It is a function of the customers' expectations that is measured by the customer satisfaction index, and of the expected revenues of company during the time period
	Value	Difference between revenues and costs, increased by the relationship value that is expected to be captured in the next time period ([26])
	Rate of Return	Ratio of the amount of money gained or lost in an investment relative to the amount invested
	Return on Equity	Ratio of the profits of a company relative to the shareholders' equity
Customer	Product Quality	Measures the performance of the product. It is a function of reliability (failure rate, mean time between failures), serviceability (mean time to repair) and conformance to customers' expectations
	Product Variety	Measures the degree of flexibility offered to the customer
	Brand Recognizability	Measures the identity of a company
	Satisfaction (as introduced in [26])	A measure to quantify the level of relationship between the company and its customers during a period of time. Satisfaction is a function of the price, delivery time, brand name, product variety and product quality
	Delivery Time Performance	Percentage of orders that are fulfilled before or at the requested date
	Price Impact on Revenues	Price change's impact on the fluctuation on total revenue amount
Operational	Process Cycle Time	Duration of business processes
	Process Cost	Cost of business processes

Context	Quality Attribute	Definition
	First Pass Yield	Percentage of products/services which could be processed without rework
	Cost of Goods	Direct cost of material and labor to produce a product or service
	Value Added Productivity	Direct material cost subtracted from revenue dividing the result by the number of employees
Learning & Growth	Investment Rate	Rate of new investments
	Innovation Rate	Ratio of the revenue with new or improved goods or services relative to the total revenue
	Training Hours	Ratio of training hours compared to overall working hours
	R&D Investment	R&D costs as a percentage of revenues

Table 3.15: Classification of KPIs

Relevance for S-Cube

KPIs are relevant to the S-Cube Quality Model as they also define quality characteristics for SBAs. They do this however on a higher level than quality models for Web services. KPIs assess the quality of an SBA on the business level, i.e., e.g. how an SBA (with its QoS characteristics such as availability and response time) affects customer satisfaction and revenues. By integrating QoS and QoBiz characteristics into a single quality model and modelling their relationships, one will be able to reason about their dependencies (i.e., how in a particular SBA, QoS characteristics affect QoBiz and vice versa) during quality negotiation and quality assurance.

Goal	Relevance	Rationale/Explanation
Quality Definition	+	The SN quality model is specified in terms of a value calculation based on KPIs. Apart from examples, there is, however, no formal model, classification or guidance yet on how KPIs are to be specified.
Quality Negotiation	-	There are no techniques to negotiate quality.
Quality Assurance	+	The quantitative analysis method as presented in [26] enables to calculate the value of the SN as a whole and of each single partner in the SN and is the basis for optimization purposes, such as outsourcing decisions.
Discipline(s)	Business Process Management	

Table 3.16: Potential Relevance of Service Networks for the Goals of WP-JRA-1.3 (+ = fully relevant, o = partially relevant, - = not relevant)

3.7 Grid Quality Model (Grid Computing)

At the time of writing there is no accepted reference quality model for Grid computing. Therefore, this section presents quality attributes currently offered by and proposed for Grid computing and combines them into a quality model. There are many different interpretations of what Grid computing is, here it is defined as a type of distributed computing infrastructure, possibly geographically-distributed and cross-organisational. This infrastructure is constructed using general-purpose software libraries and/or middleware such as Globus, Unicore, gLite or NAREGI, deployed on loosely-coupled heterogeneous computers and used to complete computationally intensive tasks and/or data storage.

A quality model is a formal description of non-functional properties of a composite system or individual system within a composite system. With the above definitions in mind, this section reviews the qualities provided by current production Grids and experimental and prototype Grid middleware. The

review is structured using the quality characteristics categories from ISO9126 [70], presented earlier in Section 3.1.

Description of the Model

Performance I: Best Efforts:

Production Grids, such as the UK's NGS (National Grid Service) [93], US TeraGrid [124], Scandinavian NorduGrid [97] and pan-European LCG (Large-Hadron Collider Grid) and EDG (European Data Grid) [77, 41], have been in operation for several years and often provide a best efforts service to their users. Generally, the performance characteristics of these Grids is known in advance and is fixed. When submitting jobs to the Grid users are often allowed to specify what configuration of hardware (e.g., number and type of CPUs and amount of memory) they would like the job to be run on so they are guaranteed a certain level of performance in a heterogeneous environment. The time taken to execute an application by the Grid is often not guaranteed because of three factors; first, production Grids are often operated under a high utilisation scheduling policy that optimises resource usage rather than task turnaround time. Secondly, the task may be user-supplied code that has not been benchmarked or run on the execution platform before, i.e. there is no historical information on how the application performs — therefore the performance and resulting execution time of an application cannot be guaranteed. Finally, even with standard, well-known applications or simulations the time required to complete a task using it may vary depending on the input parameters given to the application (as demonstrated in the butterfly effect [82]). Thus, in the quality characteristic of time to complete a job, it is common for this not to be guaranteed.

The total time to complete a task by a Grid is the sum of the time a job is queued for and and its execution time. As with the execution time discussed above, it is common for current production Grids not to commit to a queueing time. For example, on the University of Manchester's NGS Data Node "job turnaround cannot be guaranteed, as the system may be heavily utilised and incoming jobs wait in queues waiting for existing jobs to finish" [45]. Also, jobs that have been queueing longer often are given priority over urgent jobs, as this quote from a Service Level Description of a Grid cluster shows: "jobs that have been queuing for a long time are given a higher priority" [62]. Therefore, if an urgent job is submitted may be no guarantee it will be run first. Further, Grids using the popular Condor middleware, often used to backfill spaces between higher priority jobs to increase utilisation, also cannot guarantee the scheduling time (and therefore completion time) of jobs because backfilling is opportunistic and carried out whenever there is sufficient slack in the schedule.

Throughput is another measure of performance. However, because of the heterogeneous nature of jobs being executed by a Grid it is also difficult to provide a meaningful figure for the throughput of jobs (defined as the amount of completed jobs per unit time) as this will vary depending according to the number and types of jobs completed by a Grid.

Performance II: Guaranteed Resources & Completion Times:

Newer, experimental Grid middleware such as those provided by Akogrimo, AssessGrid, BREIN, BEinGRID, NextGrid and TrustCOM seek to bring a solution to the problem of meeting a guaranteed completion time (and to some extent throughput) through two techniques. The first technique is allowing the advance reservation of a period of time on a computational resource and the user to submit the job a before the period starts. Advance reservation like this is currently being retrofitted to some of the production Grids introduced above to support the requirement for co-allocation (when two systems need to work together at the same time, for example when coupling atmospheric and oceanographic simulations) and/or interactive jobs. The HARC middleware [84] has been developed to provide this function for production Grids as it allows an agreement to be made with a service provider for a guaranteed amount of time a Grid component (e.g., application, processor, storage or network) will be available for sole use. Thus, queueing time can be reduced but there is still the possibility the task cannot be scheduled at a particular moment in time or in an ad-hoc fashion because the component may already have been reserved or be in use.

The second technique to allow a Grid to meet a guaranteed deadline is to vary the amount of resources allocated to a particular task. For example, in the BREIN project each task is launched in a virtual machine running on a hypervisor platform. The progress of the job running in the virtual machine is monitored and, if the job is not meeting the agreed level of performance, more system resources (e.g., memory or share of the processor) is allocated to the virtual machine so the job completes in a shorter time. Alternatively, if the task is completing quicker than anticipated system resources can be taken from the virtual machine so it slows down.

Dependability:

In production and experimental Grid systems, monitoring systems[3] such as Ganglia [50], the Globus MDS [52] and Condor [30] are used to observe the current status and availability of Grid resources. The results of the monitoring, such as capacity of a Grid (e.g., number of hosts and/or available processors), the uptime, downtime and average reliability (jobs completed versus submitted) is made public, such as shown on the NGS Grid Reporting Page [51], the ALICE (A Large Ion-Collider Experiment) Grid status page [5] or the NorduGrid ARC monitor [98]. The non-core services such as Grid monitoring systems can also have their own qualities of service guaranteed, such as the frequency of monitoring and the freshness of monitoring data.

The experimental AssessGrid middleware builds on top of information produced by monitoring systems (in this case the OpenCCS monitoring software developed by Paderborn Centre for Parallel Computing [106]) through the mining of historical information about a Grid to give it a 'confidence' rating. Thus, the AssessGrid "middleware adds a 'risk algorithm' to the architecture of a grid, enabling [them] to calculate security, trustworthiness and dependability" [8]. Then, when a user is looking to submit a job she can use a broker that collates the confidence rating of each provider which can complete her job and use this information when deciding where to submit it.

The reasons for faults in a Grid environment can be manifold; the geographically widespread nature of a Grid which spans multiple autonomous administrative domains, variations in the configuration of different systems, resources that have unpredictable behaviour, problems with the connecting infrastructure or systems just running out of consumable resources (i.e., memory or disk space) are some of the possible sources. When discussing dependability of a Grid how it recovers from one of these faults is an important factor. Thus, automated procedures may be in place to recover from faults. For example, a Grid may try to retry the job (possibly on an alternate resource) or restart the job from a checkpoint of the application taken before the job failed. Such procedures are available when using the Taverna workflow scheduling system [122] or Condor [31], for example.

When a fault cannot be automatically fixed, production Grids often offer a mechanism for fault escalation, which can be given a priority and category definition. A good example of this is the UK's NGS service that defines categories of supported software, such as Category 1 (high-quality, well documented and understood software) to Category 3 (software provided with no promise of technical support). Faults with each of these categories and/or the Grid fabric can be assigned a severity definition, or a priority level. These levels range from Level 1 (a problem that effects most users across the entire NGS or a whole site) to Level 4 (a problem where the impact is confined to a single individual). Escalation mechanisms and procedures are then assigned to each category and priority. For example, a Category 1 problem would be dealt with by the Service Manager rather than the usual helpdesk staff.

Security:

Most (if not all) Grids provide some form of access control to their infrastructure, the first step of which is authenticating users to establish their identity. In Grid computing, this is usually carried out using public key cryptography (or asymmetric cryptography) offered through a Public Key Infrastructure (PKI) with the keys issued by Registration Authorities (RAs) on behalf of Certificate Authorities (CAs).

Each CA usually has a policy regarding the level of proof a person has to give in order to establish their identity and be issued a certificate which will represent them on the Grid. In the case of the NGS, the proof one must present in person to the RA is your passport. This should be compared with the

[3] A survey of Grid monitoring systems and a discussion of how they can be classified is given in [133].

SWITCHslcs CA (The Swiss Education & Research Network CA) that requires identity documents to be submitted remotely by fax or email [128]. Thus, certificates issued by different CAs using different policies will have different levels of assurance as to the identity of the entity using it.

Different Grids can also offer different levels of security quality based on the length of the keys issued on behalf of the CA (as information encrypted with a longer key length is more difficult to decrypt). For example, the recommended length of user and host keys is on the EDG is 1024 bits [72] whilst on the Deutsche Forschungsnetz Grid (DFG) the minimum key length is 2048 bits [38].

In addition to key length, another quality of PKI-issued keys is the lifetime of a key, after which the key will not be accepted by a Grid. The argument is the shorter the lifetime of a key the more secure it is as there is less time for it to be compromised before it is renewed. Therefore some Grids (such as the UK e-Science Certificate Authority that issues internationally-recognised keys to academics in the UK) use key lifetimes of 1 year whilst others may offer longer or shorter lifetimes.

Once a user's identity has been authenticated by their key they must be authorised when they attempt to perform actions. It is general policy that PKI keys should not be shared between users or hosts and, thus, they provide a strong assertion that the entity presenting the key is the entity the key was issued to. Thus, non-repudiation is a built-in function of such PKIs and provides a method of auditing and accounting and tracing who did what on a Grid. A common method of auditing is to analyse the log files produced by the underlying schedulers which record the start and stop times of jobs, on which node they were run and which user's identity they were run under. These logs can then be broken into Grid Usage Records (URs) [115] which form the basis for the auditing.

Logging of jobs submitted to and completed by a Grid is not the only information collected about the Grid's security environment. For instance, a CA will also log events such as certificate requests from users, acceptances of requests by the RA, certificates issued and revocation requests from users and RAs. This is to detect any strange behaviour from a particular party attempting to breech the Grid security by obtaining multiple or counterfeit certificates.

Data-related:

As well as providing a compute facility, Grids also provide facilities to store data for processing or reference. Local data written to disk may be offered with a quality of service describing the performance of the storage subsystem (i. e. speed of data read or written). In addition local or remote data can be guaranteed a level of redundancy or robustness through the capabilities of the storage subsystem. An example of where this can be found is in the HPC4U middleware [67] that can guarantee levels of resilience, such as mirroring and RAID5, and provide guarantees on the average performance of the local storage subsystem in reading and writing data [68].

On-line data will often be subject to a backup and retention policy. For example, a full backup of data in user areas may be performed every week with incremental backups taken every night, as in the HPC*x* backup policy [66]. Data stored on scratch disks may not have any retention policy at all. Backed-up data may copied to more than one replica and stored off-site for greater protection against a catastrophic data-centre failure.

Configuration management:

Grids attempt to provide a stable and predictable environment for their users and therefore many have a minimum software specification a compute or storage node must conform to. For example, [104] documents the components required for a resource to become part of the UK National Grid Service. Users are informed of any changes to components through mailing lists and on central websites well in advance (typically weeks ahead) of the work being carried out [58]. When changes are carried out the upgrades and modifications are documented in "detailed change control logs" [58]. Thus, the presence of clear change management procedures on a Grid is an indication of how stable the environment many be.

However, given the number of nodes a Grid may contain it can be the case that some computers may be out-of-step with the most recent version of an application of library when it is upgraded. Thus, many production Grids have a set of confirmation scripts and tests which can be automatically and periodically

run against each node to determine its current configuration and alert users and administrators a node does not meet the current specification. These tests can not only determine if a Grid middleware component has the correct version but also that it is correctly configured, running and capable of being used. The results of such tests can be seen at [95] and [94], for example.

Cost:

Very few (if any) production Grids have a dynamic charging capability. Groups of users (e.g., a research project or University department) are often provided time on a computing system through a research grant or centrally-administered and given a quota of time on a Grid they can use for a week, month or calendar year or the duration of the project. Thus, the economic model they operate is a token-based, fixed-cost model regardless of the actual cost of operating the Grid nodes (as some may be more expensive than others).

With the advent of experimental market inspired [19] or business-oriented Grids (BOGs) such as those provided by Akogrimo, BREIN and NextGRID the real cost of providing a service has moved to the forefront of Grid quality characteristics as a means of differentiating between services (e.g., a faster service provided in working hours may be more expensive than a slower service provided overnight) and fairly recompense Grid providers. Many different economic models have been proposed and/or implemented to determine the price of a Grid resource [19] and many more will no doubt be found in the coming years especially with the recent advent of cloud computing [23], which seeks to bring more dynamic behaviour to distributed computational computing through provisioning resources on demand and reduce the cost of operating large-scale systems [11].

Network and Infrastructure-Related:

Grid computing nodes and sites are connected through a network infrastructure. These network paths may also have QoS guarantees associated with them. Circuit-switched networks, including ATM (Asynchronous Transfer Mode) technology, can provide guaranteed and predictable data transfer rates and delays because each circuit is dedicated to one use. The UK ESLEA project [42] explored how Grid nodes could be connected using a circuit switched network (in this case provided by the UKLight service [127]) and users were able to reserve in advance a fixed amount of bandwidth, an upper-bound on latency (the delay in sending a particular packet) and jitter (the variation in latency between packets) between two points.

However, in packet-switched networks it is typical not to guarantee the precise amount of bandwidth of a link or its latency as these may vary during the period the users data is being transferred because of other traffic using the same network and infrastructure. Therefore, QoS are usually are given as a guideline or prediction from aggregated historical information. For example, providers will usually state a network link can provide a bandwidth of 'upto 2Mb/s', with no guarantee this will be met.

Usability:

Usability is a quality not often stated as a requirement in a production Grid and the lack of usability has often been a source of complaint from their users [27, 10]. When interacting with the middleware Grid users are expected to have some knowledge of the UNIX operating system, the middleware the Grid is using, general programming techniques [107] and how a PKI infrastructure (and associated tools) operate. Some relief for non-technical users has come through the deployment of Grid portals, often built using the Gridsphere toolkit [54]. These portals increase the usability of Grids for users as they give a web-style working environment for the (possibly) heterogeneous infrastructure making up the Grid and provide insulation from the workings of the middleware. Because of their ease-of-use, Grid portals play an important role in Grid interoperation as well as usability — different production Grids may be available from a single portal opening up Grid computing to people who are not computer scientists.

Regarding the functions Grid portals, typically a portal is capable of offering resource discovery (showing the status of all the clusters), job submission (possibly with easy access to popular applications) with automatic resource selection, file transfer, management and editing (e.g., create and delete and rename files, change permissions, etc.) and visualisation of results.

The new-generation of Grid middleware from the EC FP6 projects introduced above has introduced

different usability qualities to Grid. For example, in AssessGrid the Broker and Confidence services can be integrated into a portal to guide the user when making a decision about which resource provider to submit their job to.

The list of quality attributes is summarised in Table 3.17.

Context	Quality Attribute	Description/Definition
Performance		
	Scheduling strategy	How jobs are prioritised for execution.
	Hardware profile	Minimum configuration guaranteed, 'best efforts', etc.
	Data subsystem performance	Speed of local disk read and write on the Grid node.
	Job queueing time	Period of time between submitting and executing a job.
	Job execution time	Length of time required to complete a job.
	Job execution time estimation	How an estimated execution time is calculated (e.g., from historical information, simulation, on-line learning, etc.)
	Guaranteed job start/end time	How the job start/end time is guaranteed (e.g., advance reservation, priority-based).
Dependability		
	Node & job Monitoring	Monitoring policy for jobs and nodes.
	Monitoring frequency	Period of time between collection of monitoring information.
	Monitoring data freshness	How recently was the data collected.
	Service confidence rating	How 'confident' the provider or third-party broker may be that a service can meet QoS claims.
	Action on job on failure	E.g., retry, restart job from checkpoint.
	Action on node failure	E.g, Retry job on another node, restart from checkpoint.
	Fault escalation mechanisms	If job cannot be retried or restarted, how is the fault escalated?
Security		
	Credential issuing policy	Under what conditions credentials will be issued.
	Credential strength	E.g., length of X.509 key issued.
	Credential lifetime	Length of time a credential is valid for.
	Revocation policy/speed	How long it takes to get a credential revoked.
	Logging policy	Which/what security-related events are logged.
Data		
	Level of hardware redundancy	E.g., what level of RAID on-line data is stored.
	Backup scheme	How on-line data is backed up and taken off-line.
	Backup frequency	Period between full and/or incremental backups.
	Data retention period	Period of time backups are held.
	Backup replica policy	Number and type (e.g., disk or tape) of replicas taken.
	Replica storage	On-site or off-site, etc.
	Data restoration policy	How user data is restored from replicas.
Configuration management		
	Change management procedures	What are the documented change management procedures.
	Change log management	How is information about changes made disseminated and stored.
	Monitoring of node configuration	Is the configuration of nodes monitored (and/or, how?).
Cost		
	Price Model	How a price for a service is determined.
	Economic model	Micro- or macroeconomic principles for resource management (e.g., auction, bidding or commodity model).
	Charging model	Upfront, pay-as-you-go, credit-based, etc.
Network and Infrastructure-Related		
	Bandwidth	Rate of data transfer.
	Latency	Delay or wait time for data to be transferred to another node. May be one-way or round-trip.

Context	Quality Attribute	Description/Definition
	Latency variation (jitter)	Variation in latency.
	Packet loss and recovery	Packet lost on network and how lost packets are recovered.
Usability		
	Grid portal	Single interface.

Table 3.17: Grid Quality Characteristics Categorised According to the ISO-9126 Quality Dimensions

Relevance for S-Cube

From the details given above, non-functional qualities that may be guaranteed by a Grid infrastructure can be extracted. These are described according to their contexts in Table 3.17.

The Grid quality characteristics derived from the above analysis are relevant to the S-Cube Quality Model as they define quality requirements for services at the infrastructure-level. The Grid quality model presented here can be used in the negotiation of quality assurance and negotiations These quality characteristics can also be applied more generally to composite services and service-based applications.

Goal	Relevance	Rationale/Explanation
Quality Definition	+	As described in the text, the terms in this quality model are well defined and used in current Grids.
Quality Negotiation	+	The negotiation and agreement of quality characteristics have been well-studied in Grid computing. This work can be re-used in S-Cube.
Quality Assurance	o	Quality assurance methods are also well-studied, but their practical usage requires more research.
Discipline(s)	Grid Computing	

Table 3.18: Potential Relevance of Grid Quality Model for the goals of WP-JRA-1.3 (+ = fully relevant, o = partially relevant, - = not relevant)

3.8 Summary

Table 3.19 summarizes the differing relevance of the analysed quality models for the key goals of the workpackage.

	Software Engineering				SE / SOC		SOC	BPM	Grid
			UML						
Goal	**ISO**	**Reliab.**	**SPT**	**QoS**	**Infer.**	**Contr.**	**Compos.**	**SVN/KPI**	**Grid**
Quality Definition	+	o	+	+	+	+	+	+	+
Quality Negotiation	-	-	-	-	+	-	o	-	+
Quality Assurance	o	o	+	-	+	+	+	+	o

Table 3.19: Summary of Relevance of Models for the Goals of WP-JRA-1.3 (+ = fully relevant, o = partially relevant, - = not relevant)

As can be seen from the table, the different quality models are of different relevance for the goals of this workpackage. For example, where many of those models obviously support quality definition, only a few are suitable – as they are – to be used for quality negotiation.

In addition, the quality models analysed in the previous sections typically considered a small number of quality categories and in each category only some representative quality attributes. Further, these quality attributes were specific to the respective disciplines or layers of a service-based application.

Chapter 4

The S-Cube Quality Reference Model

To build the S-Cube Quality Reference Model and to address the problems identified in Section 3.8, we took into account the following considerations:

- Based on the analysed models from all relevant S-Cube disciplines, the relevant quality attributes of *all layers of a service-based application* are considered.
- We do not consider only *domain-independent* quality categories and attributes but also some of the most frequent *domain-dependent* ones like the quality categories of *Data-related*, used for services operating on and/or producing data, and *Quality of Use Context*, used for context-aware adaptive services.
- We consider quality categories and attributes that are relevant not only for the service and its service provider but also for the service requester (cf. [108]). For example, the *dependability* quality category is important for the service provider but not for the requester while the *usability* quality category is important only for the requester/user. Thus, we take into account both the *service provider* and *service requester views*.
- In each category there is an extensive list of the most representative quality attributes including not only *atomic* but also *composite* quality attributes produced from atomic ones like *response time*, *failure semantics* and *robustness*.

4.1 Quality Categories of the Model

In the remainder of this chapter, the S-Cube QRM will be described by focusing on each quality category in order to justify why we have included it, explain what is its purpose and describe some of its representative quality attributes.

We are going to present the quality attributes of the QRM in Section 4.2 in form of a table. In this table, each row will correspond to a quality attribute and will contain three fields/columns, namely *Name*, *Parent* and *Definition*. The first and the third field correspond to the name and to a small definition of the quality attribute, respectively, while the second field signifies the parent quality attribute if it exists or otherwise the category of this attribute.

Finally, a graphical representation of our service quality model is given in Section 4.3.

Performance

The *Performance* quality category contains quality attributes that characterize how well a service performs. Two quality attributes with a very well defined meaning are common among all research approaches: *response time* and *throughput*. In our quality model, *response time* is regarded as a composite

quality attribute computed from *latency* and *network delay*. Similarly, *latency* is composite and is computed from *execution time* and *queue delay time*. Finally, a quality attribute that has similar meaning with *execution time* is *transaction time* but is used in a different context (transactional services).

Dependability

Dependability of a computing system is the ability to deliver service that can justifiably be trusted [9]. In the work of Avizienis et. al. [9], the phrase "justifiably trusted" is translated into three different quality attributes and views: *availability*, *reliability* and *security*. In our opinion, *security* is orthogonal to *dependability* and must be put in separate category because it provides the mechanisms that can possibly avoid a specific type of failures from happening but it has nothing to do with the way the service has been designed and built (with respect to its proper functioning). Moreover, security mechanisms can be broken so even these faults cannot be prevented. Thus, we believe that dependability contains *availability*, *reliability*, *failure semantics* [75] and *robustness* [117, 24, 76] with the latter two attributes describing: a) the type of faults that can be exposed and how the service reacts to them (the first one) and b) the capability of the service to behave in an acceptable way when these faults happen (the second one). Another important remark is that besides *availability*, another quality attribute with similar meaning that should be added in this quality group is *accessibility* [126] as it can characterize the case where a service is available but not accessible to some users e.g. due to network connection problems.

Security

Services should be provided with the required *security*. With the increase in the use of services which are delivered over the public Internet, there is a growing concern about security. The service provider may apply different approaches and levels of providing security policy depending on the service requester. Security for services [79, 117, 76] means providing *authentication*, *authorization*, *confidentiality*, *traceability/auditability*, *accountability*, *data encryption*, and *non-repudiation*. Besides these classical quality attributes, we have added two more, namely *safety* and *integrity* [9].

Data-related Quality

In specific application domains, services do not only accept input parameters but also input data and they may also produce output data. For example, a *credit card* service can accept as input a data file describing the user's credit card information and can produce as output a data file describing details of the transaction executed based on the functionality of the service. These input/output data are characterized by quality attributes that have been traditionally used in the information and data quality domains like *accuracy* and *timeliness* [24]. Except from traditional data quality attributes, we have added two more attributes that characterize the way the service behaves with respect to the data it operates on or produces when it fails (*data policy*) and the degree of validity of the data (*data integrity* [24]).

Configuration-related Quality

This quality group/category contains quality attributes that influence the way a service is configured to function (*service level* [118]) or characterize if the promised functional and quality level has been actually delivered during the service's lifetime period (*completeness*, *stability*, *reputation*).

Network- and Infrastructure-related Quality

The network is usually used for sending requests and receiving (either instantaneously or continuously) the results back and connects the service with the requesting user. Initially, most of the research approaches [118, 6, 79, 117] were neglecting this quality aspect but after the work published in [125], this situation has changed [24, 76]. Network parameters influence the values of service quality parameters

of other quality groups like *response time* and *availability*. We have identified four network quality parameters, that are common among all research approaches, namely: *bandwidth*, *network delay*, *delay variation* and *packet loss*. Another entity that is different from the service or the user but it influences directly or indirectly service quality is the infrastructure. This entity characterizes the service execution environment and can be characterized by many quality attributes. For the time-being, we have only identified three of them, namely: *server failure*, *guaranteed messaging requirements* and *security level*.

Usability

Usability [29] collects all those quality attributes that can be measured subjectively according to user feedback. It refers to the ease with which a user can learn to operate, prepare input for, and interpret the output of the service. This quality group contains three composite (that can be further decomposed) and two atomic quality attributes. The definition of these attributes is given in the table.

Quality of Use Context

Services can become adaptive if they can change their configuration, behaviour and appearance based on the context, where "context is any information than can characterize the situation of the entity. An entity is a person, place or object that is considered relevant to the interaction of a user and an application including the user and the application themselves" [37]. So based on this definition, which is quite general, context is any information that characterizes the service and its user, their physical and execution environments (including the devices used) and the network that connects them. Context information has also quality [53, 21, 120] as it depends on the way it is sensed or derived, the time that it is produced and delivered, the level of detail and other factors. Thus, adaptive services should be designed and executed taking also into account the quality of the context that is delivered to them so as to be able to make rational and realistic decisions when to adapt and how. After reviewing the related literature in quality of context, we have identified seven (7) quality attributes from which the most important ones are: *precision* (how precise is the information), *resolution* (the level of detail), *probability of correctness* and *freshness* (age of the information).

Cost

Some research approaches consider cost as a service attribute that is orthogonal to the service quality because it is related to both functional and non-functional service attributes. However, the majority of research approaches [117, 25, 126, 24, 76] considers cost as a service quality attribute. In addition, all research approaches, at least the ones we have studied, use cost at the service selection phase in order to select the best service according to its QoS and cost and user's preferences and budget. Based on the above reasons, we regard cost as a (composite) quality attribute (and group) consisting of three (atomic) service attributes: *cost model*, *fixed costs* and *variable costs*. Actually, cost can be computed either from all atomic cost attributes or only from the *fixed costs* attribute.

Other

This quality category has been created to contain various quality attributes of services that do not belong to any other category. So the contained quality attributes may not be related to each other. For the time being, only one quality attribute has been considered called *supported standards* [6, 79, 117, 126, 24, 76] used to indicate if the service complies with standards or not. This attribute can affect the portability of the service and its inter-operability with other services or applications.

4.2 Quality Attributes

Table 4.1 provides a list of all quality attributes of the QRM and their definitions.

Name	Parent	Definition
Response time	Performance	The amount of time needed to complete a service request from the user's point of view
Latency	Response time	Time passed from the arrival of the service request until the end of its execution/service
Throughput	Performance	The number of completed service requests over a time period
Execution time	Latency	Time taken by a service to process its sequence of activities
Queue delay time	Latency	Wait time for a service request before it is actually executed
Transaction time	Performance	The time that passes while the service is completing one complete transaction
Scalability	Dependability	The capability of increasing the computing capacity of service provider's computer system and system's ability to process more operations or transactions in a given period
Capacity	Dependability	Maximum number of concurrent requests that the service is able to manage
Reliability	Dependability	The ability of a service to perform its required functions under stated conditions for a specified period of time. It is the overall measure of a service to maintain its service quality
Availability	Dependability	Availability of the service provided to customers. This is the degree of availability of the service relative to a maximum availability of 24 hours, seven days a week.
Continuous availability	Dependability	It assesses the probability with which a client can access a service an infinite number of times during a particular time period. The service is expected not to fail and to retain all state information during this time period
Accessibility	Dependability	It defines whether the service is capable of serving requests. Note that while many services are ready to use, they might not be accessible to specific clients. For instance, the connection between the service and the client is problematic or the service has already reached its threshold
Failure semantics	Dependability	They describe the general capabilities of a service to handle failures. In particular, they describe the circumstances of service failures and how a service reacts to failures
Operation semantics	Failure semantics	They describe how requests are handled in the case of failure. We can specify that issued requests are executed *exactlyOnce*, *atLeastOnce*, and *atMostOnce*
Failure masking	Failure semantics	It is used to describe what kind of failures a server may expose to its clients. A client must be able to detect and handle any kind of exposed failure
Failure tolerance	Failure semantics	Ability of a service to provide its functionality to clients in case of failures

Name	Parent	Definition
Compensation	Failure semantics	Actions that are needed to undo the effects of a service invocation when using stateful services. E.g., if a user orders by mistake a different good from a catalogue, then compensation includes to give the customer credit and to take back the order
Exception handling	Failure semantics	Internal activities that are performed in the case of failures during the execution of the service. E.g., if a database is down, a service might swap to another database to use it for persistency
Robustness/ Flexibility	Dependability	Refers to the capability of the service to behave in an acceptable way in anomalous or unexpected situations or when the *context* changes
Adaptability	Robustness/Flexibility	It refers to the capability of the service to dynamically modify its state and behaviour according to the context. The context may include user preferences, device and network characteristics, available user peripherals, user location and status (e.g. activity, mood), natural environment characteristics and service and content descriptions and can be expressed in parameters that vary significantly over time and space
Repairability	Robustness/Flexibility	The ability of a system and its repair actions to cope with any unexpected situation
Self-healability	Robustness/Flexibility	It is the property that enables a system to perceive that it is not operating correctly and, without human intervention, make the necessary adjustments to restore itself to normality
Recoverability/ Survivability	Robustness/Flexibility	The ability of the service to continue to fulfil its mission in the presence of attacks, failures or accidents
Resistance	Recoverability/ Survivability	The ability of the service to repel attacks
Recognition/ Observability/ Diagnosability	Recoverability/ Survivability	The capability of a system and its monitors to exhibit different observables for different anticipated faulty situations
Recovery	Recoverability/ Survivability	The ability of the service to restore essential services during attack and to recover full service after attack
Accuracy	Dependability	Defines the error rate produced by the service calculated on the basis of the expected results
Safety	Security	The absence of catastrophic consequences on the users and the environment
Authorization	Security	The process of determining, by evaluating applicable access control information, whether a subject is allowed to have the specified types of access to a particular resource like a service. Usually, authorization is in the context of authentication. Once a subject is authenticated, it may be authorized to perform different types of access

Name	Parent	Definition
Authentication	Security	Authentication is the process of verifying that a potential partner in a conversation is capable of representing a person or organization
Confidentiality	Security	Absence of unauthorized disclosure of information
Integrity	Security	Absence of improper system state alterations including accidental or malicious alternation or removal of information
Accountability	Security	The state of being accountable; liability to be called on to render an account; the obligation to bear the consequences for failure to perform as expected
Traceability and Auditability	Security	Capability of the service to be monitored and to generate in a reliable and secure way events producing an audit trail from which a sequence of events can be reconstructed and examined. Security events could include authentication events, policy enforcement decisions, and others. The resulting audit trail may be used to detect attacks, confirm compliance with policy, deter abuse, or other purposes
Data encryption	Security	Refers to the algorithms adopted for protecting data from malicious accesses. As one algorithm may be better than another one, it also reflects the efficiency of the data encryption algorithms and mechanisms
Non-repudation	Security	Methods to prove to the data sender that data have been delivered, and to prove the sender's identity to the recipient, so that neither the sender nor the recipient can deny operations of sending and receiving data
Data timeliness	Data-related	The property of information being able to arrive early or at the right time
Data reliability	Data-related	Trustworthiness of data; this depends mainly on the reputation of the provider
Data accuracy	Data-related	It is the measure or degree of agreement between a data value or set of values and a source assumed to be correct. It is also defined as a qualitative assessment of freedom from error, with a high assessment corresponding to a small error
Data completeness	Data-related	Represents the degree to which data values are present in the attributes that require them. Completeness can refer to both the temporal and spatial aspect of data quality, in the sense that completeness measures how much data is available compared to how much data should be available
Data validity	Data-related	It is the degree to which data values satisfy acceptance requirements of the validation criteria or fall within the respective domain of acceptable values. Validity criteria are often based on "expert opinion" and are generally viewed as "rules of thumb" although some validity criteria may be based on established theory or scientific fact
Data policy	Data-related	When a service fails and then restarts, the client needs to know if data returned by the service is still valid. To specify this, we need to associate data policy with entities such as return values or arguments

Name	Parent	Definition
Data integrity	Data-related	Degree of validity of data. Data integrity can be compromised by human errors, malicious attacks, intentional data modification, transmission errors, system/software bugs or viruses, or hardware malfunctions
Stability/ Change cycle	Configuration Management	A measure of the frequency of change related to the service in terms of its interface and / or implementation
Completeness	Configuration Management	A measure of the difference between the specified set of features (e.g. functions) and the implemented set of features
Reputation	Configuration Management	The reputation of a provider is a measure of its trustworthiness. It depends mainly on end users experience of using the service of the provider
Level of Service	Configuration Management	It is defined as the type of QoS commitment given to the application or user. Two types are usually utilized: *guaranteed service* or *best-effort service*
Bandwidth	Network and Infrastructure-related	Average of all the bandwidth samples gathered by probing the network service during intervals of length *T*
Network delay	Network and Infrastructure-related	The number of milliseconds spent in transit when a client and server exchange data. This includes the transit time for all packets required for a request-response transaction
Delay variation	Network and Infrastructure-related	It is the variation in the inter-packet arrival time (leading to gaps, known as jitter, between packets) as introduced by the variable transmission delay over the network
Packet loss	Network and Infrastructure-related	Average of all packet loss samples gathered by probing the network service during intervals of length *T*
Server failure	Network and Infrastructure-related	It describes the way in which a service can fail. That is, whether it will halt indefinitely, restart in a well defined *initialState*, or restart *rolledBack* to a previous checkpoint
Guaranteed messaging requirements	Network and Infrastructure-related	The ability of the service to ensure the order and persistence of the messages
Security level	Network and Infrastructure-related	Specifies whether security is ensured at message or transport levels
Learnability	Usability	Capability of the service to enable the user to learn how to apply/use it
Comprehensability	Learnability	Capability of the service to enable the user to understand whether its functionality is suitable, and how it can be used for particular tasks and under particular conditions of use
Content perceivability	Comprehensability	Capability of the content of the service to be used and understood by everyone, without any ambiguity or difficulty
Effectiveness	Usability	Accuracy and completeness with which users achieve specified goals
Operability	Effectiveness	Capability of the service to enable the user to operate on it
Navigability	Effectiveness	Capability of the service to provide a structure that can be easily explored and allows users to access all available information and resources
Efficiency of use	Usability	Resources expended in relation to the accuracy and completeness with which users achieve their goals

Name	Parent	Definition
Content accessibility	Usability	Ensuring that the content of the service can be navigated and read by everyone, regardless of location, experience, or the type of computer technology used
Satisfaction	Usability	Freedom from discomfort and positive attitudes towards the use of the service
Aesthetic satisfaction	Satisfaction	
Attractiveness	Aesthetic Satisfaction	Capability of the service to be attractive to the user
Precision	Quality of Use Context	Precision describes how exactly the provided context information mirrors the reality. Precision is specified with bounds. A GPS-receiver, for example, allows for a precision of about 4 meters, while positioning users via a cellular network like GSM reaches precision of up to 500 meters in urban areas
Resolution	Quality of Use Context	Resolution denotes the granularity of information. Consider a context provider announcing that the temperature in a certain room is 17 degrees Celsius. While this is on average true, there can be a hot toaster in one of the room's corners. But the context provider is incapable of offering information at a finer granularity due to restricted number of thermometers
Spatial resolution	Resolution	It is the precision with which the physical area, to which an instance of context information is applicable, is expressed. When context information describes one or more aspects of physical space, it is often implied that it is a reasonable estimate with regards to spatial resolution
Temporal	Resolution	It is the period of time to which a single instance of context information is applicable. Like space, a context also has a breadth of time to which it is applicable. Temporal resolution shows the best possible approximation of time at which a context was determined
Probability of correctness	Quality of Use Context	It is the probability that an instance of context accurately represents the corresponding real world situation, as assessed by the context source, at the time it was determined. There are several reasons due to which context information being provided may be unintentionally incorrect. This probability refers to the confidence of the source that the provided context information was accurate at the moment it was determined
Repeatability	Probability of correctness	It is one of the main ways to measure the "probability of correctness" of a context source and refers to the stability of the measure of specific context information
Up-to-dateness/freshness	Context	It describes the age of context information. In general, up-to-dateness will be specified by adding a timestamp to context information. Very often, it would be more interesting to know how well a formerly provided context information still accurately describes the actual situation

Name	Parent	Definition
Trust-worthiness	Quality of Use Context	Trust-worthiness also describes how likely it is that provided information is correct. In comparison to the probability of correctness, however, trust-worthiness is used by the context provider to rate the quality of the actor from which the context provider originally received the context information
Coverage	Quality of Use Context	The amount of the potentially sensed context about which information is delivered
Cost model	Cost	Defines a set of functions that transform resources (services) into costs
Fixed costs	Cost	Costs that constitute a fixed amount of the overall costs for the provision of a service
Variable costs	Cost	Costs that change during the provision of a service in addition to fixed costs
Supported standards	Other	A measure of whether the service complies with standards (e.g. industry specific standards). This can affect the portability of the service and interoperability of the service with others.

Table 4.1: List of Consolidated Service Quality Attributes

4.3 Graphical Representation of the Model

Figure provides a graphical presentation of the quality categories and the quality attributes of the S-Cube QRM, as well as their taxonomical relationships.

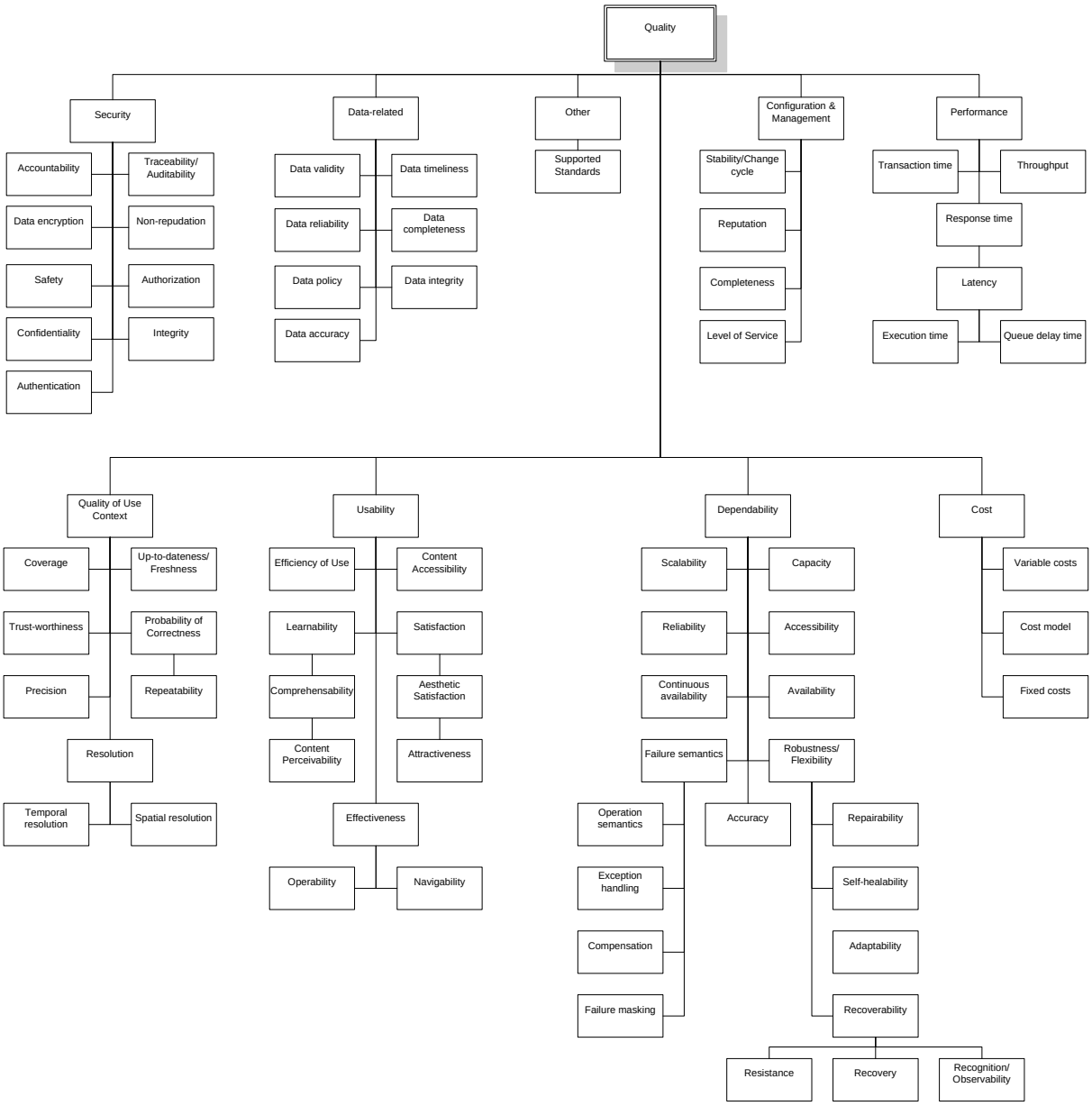

Figure 4.1: S-Cube Quality Reference Model (QRM)

Chapter 5

Conclusions

The S-Cube Quality Reference Model (QRM), devised in this deliverable, constitutes a first joint understanding of key quality attributes relevant for services and service-based applications, which have been agreed between the S-Cube disciplines software engineering, business process management, service-oriented computing and grid computing. The key definitions of the QRM have been contributed to the S-Cube Knowledge Model, where they can be accessed by a broader community of researchers.

A quality model of a service (and its infrastructure), like the QRM, is the first step for defining and enforcing service quality. The second and more difficult step is to associate the quality categories and attributes with each other, modelling in this way their quantitative and qualitative dependencies. This step is essential if we want to be able to derive more information from measurements or to evaluate the correctness of these measurements or of quality predictions.

The forthcoming deliverable CD-JRA-1.3.3 ”Initial concepts for specifying end-to-end quality characteristics and negotiating SLAs” will thus set out to devise a rich and extensible quality definition language (e.g., in the form of an extensible meta-model) that will include all relevant concepts and their relationships and inter-dependencies required for defining and monitoring end-to-end quality characteristics and negotiating SLAs.

Bibliography

[1] A. Ankolekar et al. DAML-S: Web Service Description for the Semantic Web. In *Proc. ISWC'02*, number 2342 in LNCS, pages 348–363. Springer, 2002.

[2] Jan Øyvind Aagedal and Earl F. Ecklund Jr. Modelling QoS: Towards a UML Profile. In Jean-Marc Jézéquel, Heinrich Hußmann, and Stephen Cook, editors, *UML 2002 - The Unified Modeling Language, 5th International Conference, Dresden, Germany, September 30 - October 4, 2002, Proceedings*, volume 2460 of *Lecture Notes in Computer Science*, pages 275–289. Springer, 2002.

[3] E. Albert, P. Arenas, S. Genaim, G. Puebla, D. Ramírez, and D. Zanardini. The COSTA cost and termination analyzer for java bytecode and its web interface (tool demo). In Anna Philippou, editor, *22nd European Conference on Object-Oriented Programming*, July 2008.

[4] Elvira Albert, Puri Arenas, Samir Genaim, and German Puebla. Automatic Inference of Upper Bounds for Recurrence Relations in Cost Analysis. In María Alpuente and Germán Vidal, editors, *Static Analysis, 15th International Symposium, SAS 2008, Valencia, Spain, July 15-17, 2008, Proceedings*, volume 5079 of *Lecture Notes in Computer Science*, pages 221–237. Springer-Verlag, July 2008.

[5] ALICE Job Monitoring with MonALISA. `http://pcalimonitor.cern.ch/map.jsp`.

[6] M. Anbazhagan and A. Nagarajan. Understanding quality of service for web services. IBM Developerworks website, January 2002.

[7] J. Asensio and V. Villagr. A UML Profile for QoS Management. In *Information Specification in Distributed Object-based Applications. Proceedings of the 7th Workshop HP Open View University Association, June 12.-14.*, 2000.

[8] iSGTW Feature - AssessGrid: taking the gamble out of Grid decisions. `http://www.isgtw.org/?pid=1000509`.

[9] Algirdas Avizienis, Jean-Clause Laprie, and Brian Randell. Fundamental concepts of dependability. Technical Report 0100, Computer Science Department, University of California, Los Angeles, LA, USA, 2001.

[10] B. Beckles. Re-Factoring Grid Computing for Usability. In S.J. Cox, editor, *Proceedings of the UK e-Science All Hands Meeting 2005*, pages 75–82, 2005. `http://www.allhands.org.uk/2005/proceedings/papers/565.pdf`.

[11] M-E. Bègin. An EGEE Comparative Study: Grids and Clouds — Evolution or Revolution? EGEE Technical Report EGEE-Grid-Cloud-v1.1.doc, CERN, June 2008. `https://edms.cern.ch/document/925013/`.

[12] B. Benatallah, F. Casati, D. Grigori, H. R. M. Nezhad, and F. Toumani. Developing Adapters for Web Services Integration. In *Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005, Porto, Portugal, June 13-17, 2005, Proceedings*, pages 415–429, 2005.

[13] Boualem Benatallah, Fabio Casati, Daniela Grigori, Hamid R. Motahari Nezhad, and Farouk Toumani. Developing adapters for web services integration. In *CAiSE*, pages 415–429, 2005.

[14] S. Bernardi and D. C. Petriu. Comparing two UML Profiles for Nonfunctional Requirements Annotations: the SPT and QoS Profiles. In *Specification and Validation of UML models for Real Time and Embedded Systems (SVERTS), 11. Oktober 2004, Lissabon, Portugal*, 2004.

[15] A. Bertolino, E. Marchetti, and R. Mirandola. Real-Time UML-Based Performance Engineering to Aid Managers Decisions in Multi-Project Planning. In *Proceedings of the 3rd International Workshop on Software and Performance (WOSP '02); Rome, Italy; July 24–26, 2002*, pages 251–261, 2002.

[16] A. Beugnard, J-M. Jézéquel, N. Plouzeau, and D. Watkins. Making components contract aware. *Computer*, 32(7):38–45, July 1999.

[17] Chandrasekhar Boyapati, Sarfraz Khurshid, and Darko Marinov. Korat: automated testing based on java predicates. In *Proceedings of the 2002 ACM SIGSOFT international symposium on Software testing and analysis*, 2002.

[18] BPEL v1.1. www.ibm.com/developerworks/library/ws-bpel.

[19] J. Broberg, S. Venguopal, and R. Buyya. Market-oriented Grids and Utility Computing: The State-of-the-art and Future Directions. *Journal of Grid Computing*, 2008. *In Press*. `http://www.gridbus.org/papers/MarketGridUtilityComp2001\_2007.pdf`.

[20] Antonio Brogi and Razvan Popescu. Automated generation of bpel adapters. In *ICSOC*, pages 27–39, 2006.

[21] Thomas Buchholz, Axel Küpper, and Michael Schiffers. Quality of context: What it is and why we need it. In *10th International Workshop of the HP OpenView University Association (HPOVUA 2003), Geneva, Switzerland*, 2003.

[22] Peter Buneman, Wenfei Fan, Jérôme Siméon, and Scott Weinstein. Constraints for Semistructured Data and XML. In *ACM Sigmod Record*, volume 30, pages 47–54, March 2001.

[23] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. Grid Computing and Distributed Systems Laboratory Technical Report GRIDS-TR-2008-13, The University of Melbourne, Australia, September 2008. `http://www.gridbus.org/reports/CloudITPlatforms2008.pdf`.

[24] Cinzia Cappiello. The quality registry. In Barbara Pernici, editor, *Mobile Information Systems – Infrastructure and Design for Adaptivity and Flexibility*, pages 307–317. Springer-Verlag, 2006.

[25] Jorge Cardoso, Amit P. Sheth, John A. Miller, Jonathan Arnold, and Krys Kochut. Quality of service for workflows and web service processes. *Journal of Web Semantics*, 1(3):281–308, 2004.

[26] N. S. Caswell, C. Nikolaou, J. Sairamesh, M. Bitsaki, G. D. Koutras, and G. Iacovidis. Estimating value in service systems: A case study of a repair service system. *IBM Systems Journal*, 47(1):87–100, 2008.

[27] J Chin and P.V. Covney. Towards Tractable Toolkits for the Grid: A Plea for Lightweight, Usable Middleware. UK e-Science Technical Report Series 2004-01, National e-Science Centre, 2004. `http://www.nesc.ac.uk/technical_papers/UKeS-2004-01.pdf`.

[28] Rance Cleaveland and Scott A. Smolka. Priorities in process algebra. *Information and Computation*, 87:58–77, 1990.

[29] International Standards Organisation & International Electrotechnical Commission. *International Standard ISO 9241-11 Ergonomic requirements for office work with visual display terminals (VDTs) Part 11: Guidance on Usability*. Geneve, Switzerland, 1998.

[30] The Condor Project Homepage. `http://www.cs.wisc.edu/condor/`.

[31] Condor v7.0 Checkpoint Mechanism. `http://www.cs.wisc.edu/condor/manual/v7.0/4_2Condor_s_Checkpoint.html`.

[32] J. Correas. *Analysis and Verification of Modular Programs*. PhD thesis, Universidad Politécnica de Madrid (UPM), Facultad Informática UPM, 28660-Boadilla del Monte, Madrid-Spain, June 2008.

[33] Vittorio Cortellessa and Antonio Pompei. Towards a UML profile for QoS: a contribution in the reliability domain. In *Proceedings of the 4th international workshop on Software and performance, 14.-16. Januar 2004*, pages 197–206, 2004.

[34] F. Curbera. Components contracts in Service-Oriented architectures. *IEEE Computer*, 40(11):74–80, 2007.

[35] M.A. de Miguel. General framework for the description of QoS in UML. In *Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 61–68, May 2003.

[36] N. Delgado, A. Q. Gates, and S. Roach. A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools. *IEEE Transactions on Softwware Engineering*, 30(12):859 – 872, 2004.

[37] A.K. Dey. Architectural support for building context-aware applications. Phd thesis, College of Computing, Georgia Institute of Technology, December 2000.

[38] Zertifizierungsrichtlinie der DFN-PKI v2.1, December 2006. `http://www.pki.dfn.de/fileadmin/PKI/DFN-PKI_CP_v21.pdf`.

[39] Elisabetta Di Nitto, Carlo Ghezzi, Andreas Metzger, Mike Papazoglou, and Klaus Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engineering*, 15(3-4):313–341, 2008.

[40] E. Dimitrov, R. Dumke, and A. Schmietendorf. UML-Based Performance Engineering Possibilities and Techniques. *IEEE Software*, 19(1):74–83, 2002.

[41] The European DataGrid Project. `http://eu-datagrid.web.cern.ch/eu-dataGrid/Intranet_Home.htm`.

[42] ESLEA - Exploitation of Switched Lightpaths for eScience Applications. `http://www.eslea.uklight.ac.uk/`.

[43] Huáscar Espinoza, Hubert Dubois, Sébastien Gérard, Julio L. Medina Pasaje, Dorina C. Petriu, and C. Murray Woodside. Annotating UML Models with Non-functional Properties for Quantitative Analysis. In Jean-Michel Bruel, editor, *MoDELS Satellite Events*, volume 3844 of *Lecture Notes in Computer Science*, pages 79–90. Springer, 2005.

[44] R. W. Floyd. Assigning meaning to programs. In J. T. Schwartz, editor, *Mathematical aspects of computer science: Proc. American Mathematics Soc. symposia*, volume 19, pages 19–31, Providence RI, 1967. American Mathematical Society.

[45] M. Ford. Service Level Definition for the University of Manchester Data Node on the UK National Grid Service (NGS), August 2005. http://www.ngs.ac.uk/sites/manc/NGS_Manchester_cluster_SLD.pdf.

[46] S. Frlund and J. Koistinen. QML: A Language for Quality of Service Specification. Technical Report Technical Report HPL-98-10, Hewlett-Packard, 1998.

[47] S. Frlund and J. Koistinen. Quality of Service Specification in Distributed Object Systems. *Distributed Systems Engineering Journal*, 5(4):179–202, 1998.

[48] Svend Frølund and Jari Koistinen. Quality of service specification in distributed object systems design. In *4th USENIX Conference on Object-Oriented Technologies and Systems (COOTS), April 27-30, 1998, Eldorado Hotel, Santa Fe, New Mexico, USA*, pages 1–18. USENIX, 1998.

[49] Stefano Gallotti, Carlo Ghezzi, Raffaela Mirandola, and Giordano Tamburrelli. Quality prediction of service compositions through probabilistic model checking. In Steffen Becker, Frantisek Plasil, and Ralf Reussner, editors, *Quality of Software Architectures. Models and Architectures, 4th International Conference on the Quality of Software-Architectures, QoSA 2008, Karlsruhe, Germany, October 14-17, 2008. Proceedings*, volume 5281 of *Lecture Notes in Computer Science*, pages 119–134. Springer, 2008.

[50] The Ganglia Monitoring System. http://ganglia.info/.

[51] Ganglia: NGS Grid Report. http://ganglia.ngs.rl.ac.uk.

[52] GT Information Serivces: Monitoring & Discovery System (MDS). http://www.globus.org/toolkit/mds/.

[53] Philip D. Gray and Daniel Salber. Modelling and using sensed context information in the design of interactive applications. In *EHCI '01: Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction*, pages 317–336, Toronto, Canada, 2001. Springer-Verlag.

[54] The Gridsphere Portal Framework. http://www.gridsphere.org/gridsphere/gridsphere.

[55] Object Management Group. *UML 1.5 Object Constraint Language Specification*, March 2003. Version 1.5.

[56] G. P. Gu and D. C. Petriu. Early Evaluation of Software Performance Based on the UML Performance Profile. In *roceedings of the 2003 Conference of the Centre for Advanced Studies, Conference on Collaborative Research*, pages 66–79, 2003.

[57] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[58] T. Harmer. Service Level Definition for the Belfast e-Science Centre Container Node on the National Grid Service (v0.4), April 2007. http://www.ngs.ac.uk/sites/belfast/BeSCNGSSLDv0.4.pdf.

[59] Julia Hielscher, Raman Kazhamiakin, Andreas Metzger, and Marco Pistore. A framework for proactive self-adaptation of service-based applications based on online testing. In *Towards a Service-Based Internet. Proceedings ServiceWave 2008 Conference*. Springer, LNCS, 10-13 December 2008.

[60] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10), 1969.

[61] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[62] Horace Scheduling Policy - Research Comptuing Services (The University of Manchester). `http://www.rcs.manchester.ac.uk/services/computational/Horace/HoraceUsage/HoraceScheduling`.

[63] P. Hoschka. An introduction to the synchronized multimedia integration language. *IEEE MultiMedia*, 5(4):84–88, 1998.

[64] Haruo Hosoya and Benjamin C. Pierce. XDuce: A Statically Typed XML Processing Language. *ACM Transactions on Internet Technology*, 3(2):117–148, May 2003.

[65] Haruo Hosoya, Jérôme Vouillon, and Benjamin C. Pierce. Regular expression types for xml. *ACM Transactions on Programming Languages and Systems*, 27(1):46–90, january 2005.

[66] HPCx - Data Backup Policies. `http://www.hpcx.ac.uk/services/policies/data_backup.html`.

[67] HPC4U - Introducing Quality of Service for GRIDS. `http://www.hpc4u.org/`.

[68] HPC4U Project. HPC4U - Negotiating on QoS Aspects. Deliverable D1.2 v1.3, May 2005. `www.hpc4u.eu/docs/HPC4U_D12_NegotiatingQoS.pdf`.

[69] ISO. *IS8807 : Information Processing Systems - Open System Interconnection - LOTOS - A formal description technique based on the temporal ordering of observational behavior*. ISO, February 1989.

[70] Software Engineering - Product Quality - Part 1: Quality Model. International Standard, 2001. ISO/IEC 9126-1.

[71] ITU. Specification and Description Language (SDL). ITU-T Recommendation n^o Z.100, November 1999.

[72] J. Jensen. The UK e-Science Certification Authority. In S.J. Cox, editor, *Proceedings of the UK e-Science All Hands Meeting 2003*, August 2003. `http://www.gridpp.ac.uk/papers/ah03_089.pdf`.

[73] Robert Kaplan. *The Balanced Scorecard*. Harvard Business School Press, Boston, 1996.

[74] J. Koistinen. Dimensions for Reliability Contracts in Distributed Object Systems. Technical Report Technical report HPL-97-119, Hewlett Packard, 1997.

[75] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[76] Kyriakos Kritikos. Qos-based web service description and discovery. Phd thesis, Computer Science Department, University of Crete, Heraklion, Greece, 2008.

[77] Worldwide LHC Computing Grid. `http://lcg.web.cern.ch/LCG/`.

[78] G. T. Leavens, K. Rustan M. Leino, E. Poll, C. Ruby, and B. Jacobs. JML: notations and tools supporting detailed design in Java. In *OOPSLA 2000 Companion, Minneapolis, Minnesota*, pages 105–106, October 2000.

[79] KangChan Lee, JongHong Jeon, WonSeok Lee, Seong-Ho Jeong, and Sang-Won Park. Qos for web services: Requirements and possible approaches. World Wide Web Consortium (W3C) note, November 2003.

[80] K. R. M. Leino and P. Müller. Modular verification of static class invariants. In J. Fitzgerald, I. Hayes, and A. Tarlecki, editors, *Formal Methods (FM)*, volume 3582 of *Lecture Notes in Computer Science*, pages 26–42. Springer-Verlag, 2005.

[81] Francesco Logozzo. Cibai: An abstract interpretation-based static analyzer for modular analysis and verification of java classes. In *VMCAI'07*, number 4349 in LNCS. Springer, Jan 2007.

[82] E.N. Lorenz. *The Essence of Chaos*. University of Washington Press, 1993. ISBN 978-0295975146.

[83] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, 1989.

[84] J. MacLaren, M. McKeown, and S. Pickles. Co-Allocation, Fault Tolerance and Grid Computing. In S.J. Cox, editor, *Proceedings of the UK e-Science All Hands Meeting 2006*, pages 155–162, September 2006. The HARC software is available from `http://www.cct.lsu.edu/~maclaren/HARC/`.

[85] J. Magee and J. Kramer. *Concurrency. State Models and Java Programs*. wiley, 1999.

[86] N. Mercouroff and A. Parhar. Tina computational modelling concepts and object definition language. In *IS&N '97: Proceedings of the Fourth International Conference on Intelligence and Services in Networks*, pages 15–24, London, UK, 1997. Springer-Verlag. *ISBN: 3-540-63135-6*.

[87] B. Meyer. Applying "design by contract". *Computer*, 25(10), October 1992.

[88] B. Meyer. *Eiffel the language Prentice Hall object-oriented series*. Prentice Hall, 1992.

[89] Nikola Milanovic. *Contract-based Web Service Composition*. PhD thesis, Humboldt University of Berlin, 2006.

[90] R. Milner. A calculus of communicating systems. *Lecture Notes in Computer Science*, 92, 1980.

[91] J. Navas, M. Méndez-Lojo, and M. Hermenegildo. Customizable Resource Usage Analysis for Java Bytecode. Technical Report UNM TR-CS-2008-02 - CLIP1/2008.0, University of New Mexico, Department of Computer Science, UNM, January 2008. Submitted for publication.

[92] J. Navas, E. Mera, P. López-García, and M. Hermenegildo. User-Definable Resource Bounds Analysis for Logic Programs. In *International Conference on Logic Programming (ICLP)*, volume 4670 of *LNCS*, pages 348–363. Springer-Verlag, September 2007.

[93] The National Grid Service. `http://www.grid-support.ac.uk/`.

[94] National Grid Service: Inca 2 Grid Monitoring. `http://inca2.ngs.ac.uk/conformance.shtml`.

[95] National Grid Service: Minimum Software Stack Tests. `http://inca2.ngs.ac.uk/mss.shtml`.

[96] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer, 2005. Second Ed.

[97] Nordugrid. `http://www.nordugrid.org/`.

[98] NorduGrid ARC Grid Monitor. http://www.nordugrid.org/monitor/.

[99] Object Management Group. UML 2.0 Superstructure Specification. OMG document: ptc/03-08-02, 2003.

[100] Object Management Group. UML Profile for Schedulability, Performance, and Time Specification. OMG document: formal/03-09-01, 2003.

[101] Object Management Group. UML Testing Profile. OMG document: formal/05-07-07, 2005.

[102] Object Managements Group. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, Version 1.0. formal/2006-05-02, 2006.

[103] ObjectManagement Group. UML 2.0 Infrastructure Specification. OMG document: ptc/03-09-15, 2003.

[104] P. Oliver, A. Richards, and S. Young. NGS Minimum Software Stack. National Grid Service Working Document, March 2006. https://www.ngs.ac.uk/man/documents/NGS_Minimum_software_stack.pdf.

[105] OMG. The Object Constraint Language Specification 2.0, OMG Document: ad/03- 01-07, 2007.

[106] OpenCCS - Computing Center Software. https://www.openccs.eu/core/.

[107] Oxford University Computing Services. Oxford Node on the UK National Grid Service (NGS) Service Level Description (v4.0), May 2004. http://e-science.ox.ac.uk/ngs/Oxford_SLD_v4.pdf.

[108] Michael P. Papazoglou and Willem-Jan Van Den Heuvel. Service-oriented design and development methodology. *Int. J. Web Eng. Technol.*, 2(4):412–442, 2006.

[109] Mike Papazoglou and Klaus Pohl. Report on longer term research challenges in software and services. Results from two workshops held at the European Commission premises at 8th of November 2007 and 28th and 29th of January 2008, European Commission, www.cordis.lu, 2008. With contributions from Boniface M, Ceri S, Hermenegildo M, Inverardi P, Leymann F, Maiden N, Metzger A, Priol T.

[110] Barbara Pernici and Andreas Metzger. Survey of quality related aspects relevant for SBAs. Deliverable PO-JRA-1.3.1, S-Cube, 2008.

[111] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981. *ISBN: 0136619835*.

[112] P. Pietrzak, J. Correas, G. Puebla, and M. Hermenegildo. Context-Sensitive Multivariant Assertion Checking in Modular Programs. In *13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR'06)*, number 4246 in LNCS, pages 392–406. Springer-Verlag, November 2006.

[113] Marco Pistore and Paolo Traverso. Assumption-Based Composition and Monitoring of Web Services. In Luciano Baresi and Elisabetta Di Nitto, editors, *Test and Analysis of Web Services*, pages 307–335. Springer, 2007.

[114] Shankar R. Ponnekanti and Armando Fox. Interoperability among independently evolving web services. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 331–351, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[115] R. R. Mach, R. Lepro-Metz, S. Jackson, and L. McGinnis. Usage Record Format Recommendation GFD-R-P.098. Open Grid Forum Recommendation, February 2007.

[116] P. Radha Krishna, K. Karlapalem, and D.K.W. Chiu. An ER^{EC} framework for e-contract modeling, enactment, and monitoring. *Data Knowl. Eng.*, 51:31–58, 2004.

[117] Shuping Ran. A model for web services discovery with qos. *SIGecom Exch.*, 4(1):1–10, 2003.

[118] Bikash Sabata, Saurav Chatterjee, Michael Davis, Jaroslaw J. Sydir, and Thomas F. Lawrence. Taxomomy of qos specifications. In *WORDS '97: Proceedings of the 3rd Workshop on Object-Oriented Real-Time Dependable Systems - (WORDS '97)*, pages 100–107, Washington, DC, USA, 1997. IEEE Computer Society.

[119] Sagar Sen, Benoit Baudry, and Jean-Marie Mottu. On combining mullti-formalism knowledge to select test models for model transformaion testing. In *IEEE International Conference on Software Testing*, Lillehammer, Norway, April 2008.

[120] K. Sheikh, M. Wegdam, and M. J. van Sinderen. Quality-of-context and its use for protecting privacy in context aware systems. *Journal of Software*, 3(3):83–93, March 2008.

[121] Synopsys Inc., http://www.systemc.org. *SystemC v2.0.1 Users Guide*, June 2002.

[122] Taverna Project Website. `http://taverna.sourceforge.net/`.

[123] M.H. ter Beek, A. Bucchiarone, and S. Gnesi. Formal Methods for Service Composition. *Annals of Mathematics, Computing and Teleinformatics*, 1(5):1 – 10, 2007.

[124] Teragrid. `http://www.teragrid.org/`.

[125] M. Tian, A. Gramm, M. Nabulsi, H. Ritter, J. Schiller, and T. Voigt. Qos integration in web services. Gesellschaft fur Informatik DWS 2003, Doktorandenworkshop Technologien und Anwendungen von XML, October 2003.

[126] Hong-Linh Truong, Robert Samborski, and Thomas Fahringer. Towards a framework for monitoring and analyzing qos metrics of grid services. In *International Conference on e-Science and Grid Computing*, Amsterdam, The Netherlands, December 2006. IEEE Computer Society Press.

[127] JANET Lightpath. `http://www.ja.net/services/lightpath/index.html`.

[128] A. Usai. Grid Certificates for Users in Switzerland. `https://www.switch.ch/pki/meetings/2008/switchpki_grid_certificates.pdf`.

[129] J. Vonk and P. Grefen. Cross-organizational transaction support for E-services in virtual enterprises. *Distrib. Parallel. Dat.*, 14:137–172, 2003.

[130] L. G. Williams and C.U. Smith. *Performance Solutions A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley, 2002.

[131] WSCDL v1.0. www.w3.org/TR/2004/WD-ws-cdl-10-20040427.

[132] WSMO working group. www.wsmo.org.

[133] S. Zanikolas and R. Sakellariou. A Taxonomy of Grid Monitoring Systems. *Future Generation Computer Systems*, 21(2005):163–188, July 2004. `http://www.cs.man.ac.uk/~zanikols/fgcs05.pdf`.