# D3 – Method for Service Composition and Analysis

Wolfgang Emmerich  D. Davide Lamanna
Giacomo Piccinelli  James Skene

October 21, 2003

# Contents

# Chapter 1

# Introduction

## 1.1 This document

This document describes the 'Method for Service Composition and Analysis' under development as part of the TAPAS project. It is intended for submission as deliverable D3.

## 1.2 The TAPAS project

### 1.2.1 Vision

The vision of the TAPAS project is to enable the construction of federated distributed systems with dependable Quality of Service (QoS) and security properties. In a federated system components are owned and operated by seperate commercial entities. The quality of a system as a whole depends on the qualities of its subcomponents, and so organisations must rely on contracts or trust relationships to ensure that their systems are not degraded by the performance of a component provided by another organisation. The commercial and technical context for the TAPAS project is today's internet.

Distributed applications can potentially make use of a wide variety of internet services. These include: Internet Service Provision (ISP), permitting the flow of data between network endpoints, Storage Service Provision (SSP), providing reliable data management, Application Service Provision (ASP), to outsource part of the functionality of a system, and hosting provision, to enable system components to reside at particular network locations and benefit from a standard execution environments.

Guaranteeing quality and security properties for a federated application requires guarantees of these properties for all aspects of its deployment. For the purposes of the work documented here, we have defined a reference model of distributed system architecture, show in Figure 1.1. This identifies the types of service and service interaction for which we intend to provide trust and QoS management.

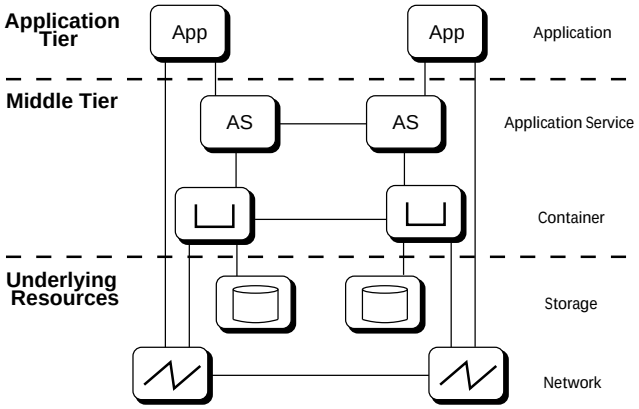


Figure 1.1: Reference Model of Distributed System Architecture

Clients of a service are vunerable to the service provider in at least two ways: Firstly, the client may depend on the capabilities of the service to conduct their business. If the service is unavailable or is delivered poorly, the client will be compromised. The client may have invested in the integration of the service, or the service may be unique, making it impossible to simply change to another provider. Secondly, the client may pass sensitive data to the service. This data may be vunerable to abuse by external parties, other clients of the service, or the service provider itself.

Service providers are also vunerable to the actions of their clients. Quality attributes such as performance often depend on external factors such as workload. If one client overuses the service, its performance might degrade for all other clients. Also, by permitting access to the service, the service provider presents additional opportunities for the client to violate the security of the service, possibly to steal information sensitive to other clients.

The TAPAS project aims to reduce these mutual vunerabilities in order to ease the construction of federated distributed systems.

### 1.2.2 Approach

The approach taken by the TAPAS project is to enable the specification of service-provision contracts between service providers and their clients, and to provide architectural support for the provision and monitoring of these services. Service agreements compel the provider of a service to meet specified levels of quality and security, mitigating the vunerability of the client to the provider. They also prevent the client from abusing the service, mitigating the vunerability of the provider to the client.

Service-provision contracts are legally binding agreements between the client and provider of a service, and their complete definition is outside the scope of the TAPAS project. However, those aspects of the agreements pertaining to the technical characteristics of the service require precise definition. To this end, a language for Service Level Agreements (SLAs) has been defined, called SLAng, described in deliverable D2 [27]. Statements in SLAng describe the responsibilities of providers and clients of an service with respect to a set of quality parameters. The qualities are predefined in the language to provide a standard basis for negotiation between clients and providers. The language is targetted at the services and interactions present in the reference model, Figure 1.1.

For service agreements to be useful it must be possible to monitor them in such a way that violations can be detected and unambiguously attributed to the responsible party. The TAPAS platform, documented in deliverable D7 [12], will provide architectural support for service monitoring.

In the face of strict constraints on quality and security properties, the development and maintenance of services becomes more difficult. The TAPAS project therefore aims to provide support for the development of secure services with managed quality properties by adapting existing middleware and networking techniques as the basis of the TAPAS platform. These will provide an implementation framework including support for deployment and adaption of applications with guaranteed QoS and security properties.

The project also aims to develop a method for service composition and analysis. This is a methodological component of the project supporting development and decision-making activities necessary for clients and providers of QoS aware services. These include: Capacity planning of services; the integration of services into federated distributed applications; the matching of required levels of service to service offerings; and the development of new applications and services with dependable qualities, using a combination of existing technologies and the TAPAS platform. The proposed method for service composition and analysis is

the subject of this document, and is introduced in detail in the next section.

Mutual dependencies exist between the technical and methodological support for the development of federated applications provided by TAPAS. The deployment and monitoring facilities provided by the TAPAS platform potentially support stronger guarantees of quality than existing technologies, and must therefore be incorporated into the model of services supporting notions of composition and analysis. Throughout this document we highlight opportunities to incorporate TAPAS platform details into composition and analysis methods, and additional opportunities for the TAPAS platform to support the service composition activities identified in the next section.

## 1.3 Method for Service Composition and Analysis

This document only deals with QoS aspects of service composition. Trust and security aspects will be dealt with in deliverable D9.

We assume: That any party wishing to reason about QoS owns, or is developing, at least one distributed system or service; that the design of that system is visible to them; that the design may incorporate the TAPAS platform; and that parts of the design may rely on services provided by external organisations. The design of external services will not necessarily be available. The method for service composition and analysis is required to answer the following questions:

1. What will the QoS properties of the system be, given a certain set of environmental factors, such as workload?
2. How do the QoS properties of the system depend on the assurances given by the external services that it integrates?
3. How should the system be developed or modified to provide dependable QoS?
4. If the system is a service, what SLAs can the application support, and for how many clients?
5. Given that the system depends on a certain level of service from its subservices, what SLAs should be specified for those subservices, or how should commodity SLAs be chosen?

In this document we outline a method for answering these questions and discuss how the method integrates with software development and maintenance

processes.

The method requires that organisations maintain system designs using the Unified Modelling Language (UML) [44]. These designs serve as the basis for analysis activities required to answer Questions 1 and 2 above. They potentially incorporate architectural components from the TAPAS project. The identification of these architectural components in the designs guides development by refinement, addressing Question 3. The designs incorporate information about SLAng SLAs governing service interactions. The comparison of these SLA terms with QoS properties identified by analysis or monitoring addresses Question 4. The comparison of required levels of QoS identified by analysis with offered SLAs, in combination with the notion of SLA combatibility defined with reference to the SLAng semantics addresses Question 5.

We employ a number of domain-specific extensions to the UML, to support the representation of architectural components, QoS properties and SLAng SLAs. By employing UML as a base language we adopt an industry standard design notation already widely employed by developers of internet services. We gain tool support for our method, allowing the production of design and analysis models. The UML models used provide a single repository for information relating to service composition and analysis, allowing centralised management of models and design information.

Throughout this document, application services are used to illustrate the approach taken. We regard the federation of application services as a prime objective of TAPAS. Application services are considerably more diverse than the other services identified in the reference model, and this makes support for service composition with dependable QoS a significant challenge. We believe that the modelling approach taken to address this challenge will also be applicable to the composition of services of the other types identified. This will be demonstrated in future work.

### 1.3.1 QoS Prediction

A means to predict the QoS properties that a system will exhibit given a particular configuration and environment is necessary because it may be impossible or impractical to test the system under the conditions of interest. It may be difficult or expensive to establish the test conditions, for example generating a realistic workload for a system intended to service millions of clients. The system may not exist as specified, the purpose of the prediction being to evaluate design decisions before committing resources to implementation.

QoS prediction requires a view of the behaviour of the system, and the QoS

properties of its components. UML models can provide such a view. Prediction may also require sophisticated analysis techniques to determine the QoS properties emerging from the interaction of many components. Previous research has proposed derivations of analysis models such as stochastic Petri-nets [51] and layered queuing networks [48] from UML designs. A UML extension, 'The Profile for Schedulability, Performance and Time Specification' (henceforth 'the real-time profile) [42] has been standardised to allow modelling of the performance properties of systems where resource utilisation and scheduling are the principle influences on performance. The intent of the real-time profile is to allow analysis techniques to use model data as input.

Previous work suffered in several respects: Firstly, UML and its extensions do not have a formally defined semantic, meaning that there can be no strong proof of the validity of a particular derivation. This is unlikely to change as it has thus far proved impossible to build a consensus for strong formality for UML. Secondly, the derivations proposed in the literature are defined using a number of ad-hoc techniques ranging from graph-grammars to natural language descriptions. The lack of a standard representation hinders their deployment, and when deployed the technique becomes coupled to a particular tool. Thirdly, a completely encapsulated derivation is unlikely to produce successful analyses every time because of the difficulty in deriving a feasible and valid model for all designs. This implies that tools are likely to be brittle if they cannot be adapted.

In Chapter 3 we describe our approach to analysing non-functional properties of distributed software architectures based on Model Driven Architecture (MDA) technologies [38]. MDA is a development approach based on UML models, in which business knowledge (Platform Independent Models - PIMs) is maintained seperately from technical artefacts, such as design models (Platform Specific Models - PSMs) and source code. The successful application of the MDA approach depends on technologies and tools supporting flexible modelling of diverse semantic domains (PIMs and PSMs), and relationships and transformations between them (deployment of PIMs to PSMs). We use the UML profile mechanism to define classes of analysis models, design models and the mapping between the two. Profiles are denoted using UML and may be injected into any conforming tool, reducing tool tie-in. The derivation is visible and modifiable within a tool, adding flexability. Its colocation with design and analysis models reduces ambiguity due to the relatively informal semantics of UML.

We provide an example of the method applied, in which queuing network models are derived from UML designs annotated using the real-time profile. We

also discuss the derivation of analysis models suitable for other types of QoS property, such as reliability. Finally, we discuss the way that analysis model derivations can capture expert modelling knowledge relating to architectural components identified in designs, for example EJB components, or elements of the TAPAS architecture. Architectural components are both an advantage and a disadvantage in performance analysis. They enable more accurate predictions because they are standard and their implementations change infrequently, making detailed modelling a safe investment of effort. However, they are also monolithic, so determining their QoS characteristics can be difficult in the first instance. It should be a goal of the TAPAS platform to provide transparent QoS characteristics to ease analysis modelling.

### 1.3.2 SLA representation and reasoning

If internet services are to deliver guaranteed service levels then their behaviour must be understood in terms of the quantities specified in service level agreements. Consequently there is the need to formally define the meaning of the SLA language used, in our case SLAng, and to establish the relationship of its parameters to the results of analysis.

We address this problem as follows: We define a formal semantic for SLAng; we define a UML extension that allows SLAng contracts and the services to which they apply to be modelled in UML designs; and we define a UML extension that allows modelled QoS properties to be related to terms in SLAng. Our approach is describe in Chapter 4.

Unambiguously defining the meaning of SLAng was a significant challenge. We use the Unified Modelling Language (UML) to model the language, producing an abstract syntax. We embed this language model in an object-oriented model of services, service clients and their behaviour. The presence of SLAs, instances of the language model, constrains the behaviour of the associated services and service clients. The constraints are defined formally using the Object Constraint Language (OCL), a sub-language of UML with an unambiguous interpretation, and with accompanying natural language descriptions. The constraints define the semantics of the language, and are easily understood in the context of the service model.

Benefits of the formal semantic include the reduction of ambiguity in the meaning of the language through the association of a reference model of service behaviour, and the means to check the semantics to ensure the absence of inconsistencies and loopholes. The style of semantic definition is user-friendly so it can serve as a reference for human negotiators. It also provides a formal ba-

sis for comparisons between SLAs, and an abstract reference model of systems employing SLAs that can guide implementation and analysis efforts.

The semantic supports the comparison of desired service levels, expressed as target SLA terms, with offered service levels. A required and an offered SLA are said to be compatible if the offered SLA permits no behaviours (according to the service model) that would violate the target SLA. Checking compatibility between SLAs is a key facility for developers who must choose external services to meet their requirements.

In order to relate SLAs to the systems they constrain, we employ our semantic model as the basis for a UML extension. This provides the additional vocabulary to the model of services and SLAs in the same context as more detailed design information. The UML extension is provided in the form of a QoS catalogue appropriate to the proposed 'UML Profile Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms' (henceforth 'QoS profile').

It is also necessary to relate the SLA parameters to QoS properties determined by analysis. Because analysis profiles such as the real-time profile are based on a different semantic model to SLAng, it is not possible to conduct analysis using SLAng properties directly. A possible solution to this is to unify semantic models. However, this would have the negative effect of coupling profiles for different purposes to an unnecessarily complex semantic model. For now, we have chosen to represent QoS characteristics alongside models of SLAng contracts, and introduced a notation allowing the designer to assert that a QoS property corresponds an SLA parameter.

### 1.3.3 Implementing QoS-aware systems

UML is the design language of choice when adopting a Model Driven Architecture (MDA) [38] development strategy. In this methodology models of business knowledge are maintained seperately from technical models and artifacts, enhancing their reusability across multiple platforms. Model transformation is a key technology in the MDA, used to deploy business knowledge automatically in new technical domains. Transformations rely on the semantics of both source and target domains.

We employ domain specific languages to represent architectural components, both as the starting point for analysis, as discussed in Chapter 3, and as points of attachment for SLA specifications when using the SLAng catalogue, presented in Chapter 4. Models including these architectural components are effectively high-level platform-independent models in the MDA terminology, and are

therefore a suitable starting point for implementation efforts, either driven by refinement or assisted by model transformation technologies. Such refinements or transformations will be informed by the rich semantic defined for SLA-aware services by the SLAng semantic model.

In the future, it may also prove beneficial to provide semantic models directly describing the TAPAS architecture, and related architectural components such as EJB containers. Such semantic definitions can serve as reference models for UML extensions [13, 50]. UML diagrams employing these extensions could therefore form the basis for more sophisticated deployment transformation respecting the platform semantics. We have investigated this approach by defining a semantic model and UML profile for electronic-service systems with workflow components. This work demonstrates the feasibility of the approach, suggesting that an application to the TAPAS project would be a useful addition to the work presented here. We review this work in Chapter 5.

### 1.3.4 Methodological Implications

The reliance of our method on UML suggests that organisations wishing to apply it will have to adopt a model-centric development process, such as the Rational Unified Process [24]. We rely only on standard UML technologies for our method, but its philosophy is based in MDA concepts. In particular, if organisations wish to gain additional benefit from the platform independent models of QoS-aware systems, architectures, services, and SLAs then they will have to adopt a refinement-based development approach, possibly structuring their systems along MDA lines.

For organisations employing a development process whose primary artifact is UML models, the overhead of our analysis approach is not dramatic. Analysis becomes possible as soon as designs reach an adequate level of detail to permit analysis models to be derived. The expense of annotating designs with QoS information is proportional to the effort expended to ensure the accuracy of that information. 'Rule of thumb' assessments of the possible performance of a design are possible quite cheaply. The derivation of analysis models and the processing of those models to produce results can and should be automated.

The use of SLAs to mediate service provision is a significant managerial decision. Service provision must be planned to meet capacity and performance targets. The SLAs themselves must be carefully parameterised. Our semantic for SLAs, in combination with our approach to analysis provides strong support for these activities. The manipulation of SLAng SLAs is supported by its machine-readable syntax and its rigorously defined semantics, which provides a

reference for negotiation, monitoring, implementation and analysis activities.

## 1.4 Overview

In Chapter 2 we provide technical background concerning the UML and MDA development approach supporting the material in subsequent chapters; in Chapter 3 we present our approach to non-functional analysis of UML designs; in Chapter 5 we present a model and profile for electronic service systems, exemplifying a suitable approach for the modelling of service systems; in Chapter 4 we present a formal semantic for the SLAng language and use the semantic model as the basis for a UML extension for modelling SLAs; in Chapter 6 we conclude and discuss future directions for this research.

# Chapter 2

# Background

## 2.1 UML

The Unified Modelling Language (UML) is an object-oriented graphical language that has found wide applicability in analysis and design for software engineering. In this document we describe how UML can be used as a basis for modelling services, their qualities and service level agreements. This modelling depends on domain-specific extensions to UML delivered using profiles. Profiles are a UML extension mechanism whereby the innate notations provided by the UML can be augmented with labels, called 'stereotypes', tagged values and constraints, which provide semantic refinement, annotations and syntactic refinement respectively.

UML is based on a conceptual architecture that is divided into four meta-modelling layers as shown in Figure 2.1. The lowest level is the data layer (M0), in which objects such as data-patterns in computer memory and other real-world phenomena including people and things are supposed to reside. The elements in the lowest level are classified by types in the UML models that analysts and designers produce, which hence reside at the next meta-level (M1). UML model elements are, in turn, objects of classes in the UML meta-model (M2). Attached to these meta-classes are semantic descriptions and syntactic constraints that control the meaning and applicability of the UML. The meta-model at level M2 is self-describing, so can also be regarded as residing in level M3 (and plausibly all higher levels).

Profiles then, are a means of refining classes, semantics and syntactic constraints at the M2 level. Confusingly, profiles are defined at the M1 level, so that they can be denoted using UML and deployed by including them with any UML

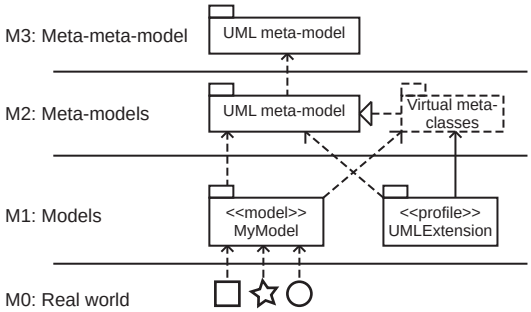


Figure 2.1: Meta-modelling architecture of the UML

model that requires their language extensions. They can therefore be regarded as injecting 'virtual meta-classes' into the UML meta-model (M2).

In its current form the UML is an engineering language, appropriate for communications between engineers. Its semantics are defined by drawing associations between model elements and real world things (such as data, behaviour, roles, events), primarily using natural language. The vocabulary used, for example 'class', 'collaboration' etc. has a common interpretation in English, and the UML specification also provides extended descriptions of these elements. Subtleties of the semantic domain are captured in the form of constraints, expressed in OCL, which prevent users from producing models that would represent illogical situations in the real world, further enforcing and disambiguating the informal semantics defined for the language. We use OCL constraints in profiles in the same way. Constraints, acting at the meta-model level (M2), ensure that the formal models produced are reasonable.

When presenting profiles, it is common to first present a domain model [13]. These are similar to the UML meta-model, and can be considered to reside at level M2 in the conceptual architecture. Domain models can be thought of as a new meta-models that describes the semantic domain directly, independently of the need to refine the semantics of the UML meta-model.

## 2.2 The Model Driven Architecture

The Model Driven Architecture (MDA) [38] is a modelling approach based on UML. It recommends that development organisations separate models of their business logic (Platform Independent Models - PIMs), from technical artifacts, such as design models (Platform Specific Models - PSMs) and source code. The

benefit is to insulate organisations from the cost of re-deploying software services as architectural infrastructures change, particularly middleware standards. The approach also supports the integration of heterogeneous and legacy software. For these reasons it is extremely well suited to development tasks in an electronic service environment.

UML has been widely adopted in industry to represent software designs, particularly those which are to be implemented in one of the currently dominant object-oriented programming languages, such as Java, C++ and C♯. However, it is its heritage as an analysis language (in the sense of requirements capture and problem-domain modelling) that makes it a suitable core representation for the MDA approach, as the ability to represent a technology-neutral PIM is provided by the facilities that support the modelling of problem domains in object-oriented analysis.

Model transformation is central to the MDA approach. Transformations from PIMs to PSMs deploy business logic on new technical platforms. If these transformations can be automated, the development organisation realises significant cost savings when redeploying the application. The OMG is in the process of standardising UML extensions for the description of transformations. However, OCL, applied at the metamodel layer using profiles can already be used to describe consistency rules between models. These consistency rules can be regarded as a contract or template for transformations. In the next chapter we apply this approach to deriving analysis models from designs. This application of transformations is unconventional from the perspective of the MDA approach, as it introduces models that are not clearly either PIMs or PSMs. These formal models can be considered to be views of the system in terms of non-functional properties.

Extensions to the UML provide the means to denote designs more concisely. By introducing domain-specific vocabulary using profiles the user can avoid the overhead of expressing standard aspects of the design in fine detail. However, this means that transformations from such designs must be informed not only by the design model, but by the semantics of the extensions used in this model. This makes the provision of domain models as a reference for UML extensions vital when supporting the MDA development approach. In Chapter 5 we show how a UML extension with an associated domain model could support the TAPAS architecture, using the example of electronic service systems, a related class of distributed systems.

# Chapter 3

# Performance Analysis of Designs

Formal analysis is often the only way to determine whether an architectural design will meet its QoS requirements, such as performance and reliability. Formal analysis is rarely performed partly because of difficulties inherent in its application, such as the need to employ unusual high-level languages, to combat state space explosion, a lack of integrated tool support, and a lack of understanding of the impact of architectural components on QoS.

Here we present our approach to delivering formal analysis in industrial design tools, specifically UML tools. The approach used the MDA principle of model transformation to derive analysis models from annotated designs. The derivation is expressed using logical constraints between design and analysis models. The constraints are captured in a profile, a semantic extension to the UML. The constraints form a contract for model transformation techniques, allowing the transformation to be automated. Both the derivation and the resulting analysis model are visible and modifiable within the tool adding the flexability required to accomodate the difficulties inherent in deriving valid analysis models for all designs. The transformation captures the expert knowledge required to produce an analysis model for the design domain. It therefore has the potential to integrate knowledge about the impact of architectural components on QoS.

## 3.1 Approach

Our approach to delivering expert analysis techniques is to specify the derivation of a formal analysis model from a UML design using logical constraints. This is similar to the derivation of a PSM from a PIM in the MDA development approach.

In our approach we define a profile which extends UML to model specifications in some formal language. This language can be used for analysis, so these specification models can be operated upon to generate results. The MDA technologies include standard interfaces for operating on model data [41, 39], so there is the potential to closely integrate model solvers with design tools.

We use a profile for the design domain to direct the derivation of analysis models. Constraints ensure that design models are reasonable and contain sufficient detail to permit a derivation.

The derivation itself is specified in a third profile, allowing reuse of design and analysis profiles. The mapping profile provides a stereotype on associations. The refined type of association is constrained to be between two sub-models, one representing the design and the other the analysis model. Additional constraints on the contents of the sub-models, ensuring that the analysis model correctly represents the design.

Profiles are repesented using UML and may be used wherever necessary by including them in a model. Therefore, profiles specified according to our scheme can be imported into standards-compliant tools to enable formal modelling and the consistent association of formal models with designs.

To provide a completely automated derivation of formal models it is necessary to implement an additional model transformation algorithm, perhaps using a tool-specific scripting language. The mapping constraints provide the contract for such an algorithm, and can be used to test its operation. Figure 3.1 shows the overall approach.

Results can also be reintegrated into design models by applying a mapping from a results domain to tagged-values or notes in the design domain. We do not consider this in this paper.

The approach is applicable to analyses relying on graphical formalisms, as UML can easily be extended to resemble these. For example, performance evaluation and functional analysis using Petri nets or Markov chains. [52] proposes to use the technique for reliability modelling, and Baysian networks or fault trees would be an appropriate formalism. Our example in the next section uses queuing networks to forecast performance and resource utilisation.

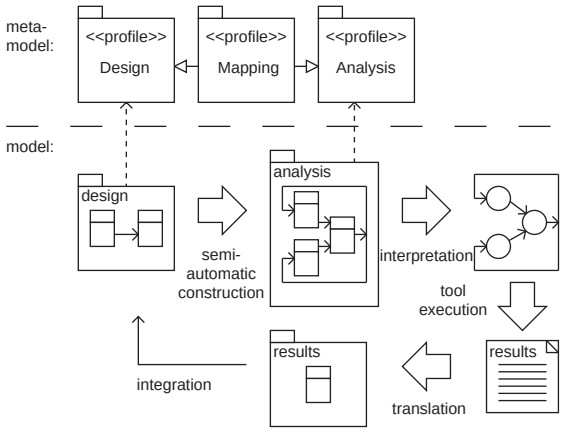


Figure 3.1: Approach

## 3.2 Example: Analysing real-time UML using queuing networks

In this section we provide an example of our approach applied to define a derivation from a class of architectural models to queuing networks. To demonstrate the way that this derivation would be used in practice, we present a running example based on a hypothetical website content-management system. The profiles and example packages are shown in Figure 3.2.

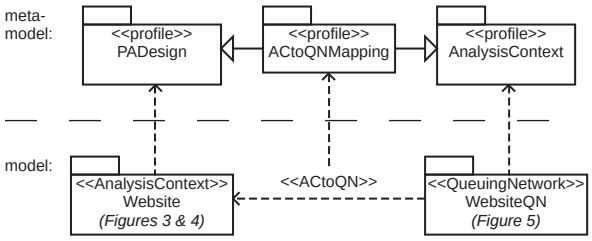


Figure 3.2: Example profiles

The design model profile closely resembles the performance subprofile of the standard Profile for Schedulability, Performance and Time Specification (the 'real-time profile') [43], which permits the modelling of systems containing contended resources. Reuse of such standard profiles enhances the applicability of our approach.

To permit analysis the designer must model the environment of the system

in terms of populations engaging in use-cases, identified by a stereotype on an Actor. In our example there is a large `<<OpenPopulation>>` of site users, characterised by an exponential arrival rate, and also a a small `<<ClosedPopulation>>` of editors who interact after a think-time.

The behaviour of the system is modelled using a sequence diagram for each use-case, including `<<step>>` actions with resource demands. The 'user' use-case is shown in Figure 3.3.

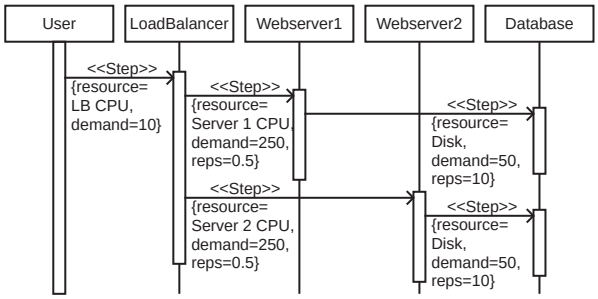


Figure 3.3: Sequence diagram, describing demands associated with a use-case

The structure and `<<resource>>`s of the system are modelled using a deployment diagram, as shown in figure 3.4

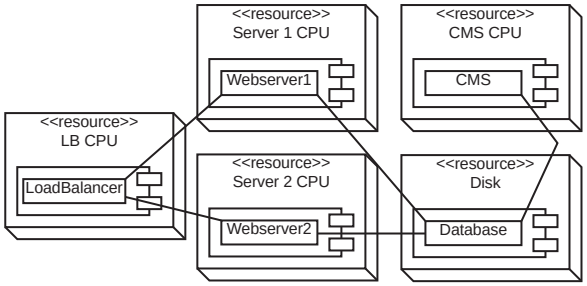


Figure 3.4: Deployment diagram, describing system resources

Profile constraints are presented here in natural language, but all have a counterpart defined formally using OCL. An example is provided later in the section. The full OCL constraints are available on the web [54].The design

profile constraints are:

1. An analysis context must contain at least one population.
2. A population (stereotype on an actor) must be associated with a use-case.
3. Every use-case must be realised by exactly one associated interaction (the dynamic part of a collaboration).
4. Each interaction should include at least one message with a resource demand.
5. Resources must be uniquely named.
6. Every message with a resource demand must be sent to a role deployed in a context where the resource is available.

Analysis models are queuing networks. Figure 3.5 shows the queuing network derived from our example design, including the demands due to editors.

Analysis model constraints:

1. The network must contain at least one workload class
2. Classifiers can be either queues or workloads, not both
3. All associations stereotyped as demands must start from a classifier stereotyped with a workload and end in one stereotyped as a queue.
4. All queues must be demanded by at least one workload class.
5. All workloads must demand at least one queue.

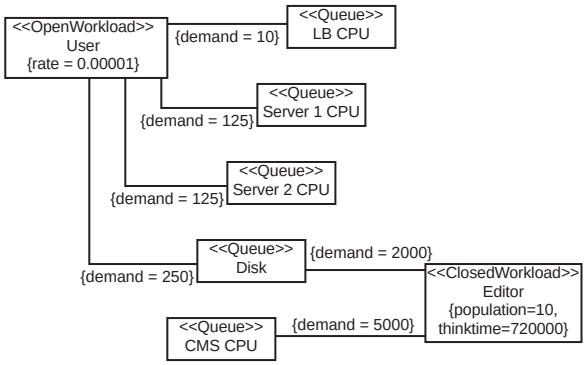


Figure 3.5: Queuing network model

The mapping is defined in the context of a stereotype `ACtoQN` defined on an association. Mapping constraints are:

1. The mapping must be between an analysis context model and a queuing network model.
2. Every population must correspond to exactly one workload class in name, type and tagged values.
3. Every workload must have exactly one corresponding population.

4. All resources with resource demands in a use-case must be represented by queues.
5. All resource demands present in an interaction must be represented by demands on associations between the corresponding workload and queue. The demands must equal the sum of the products of action demands and action repetitions within the workload.

Constraint 3 above is expressed in OCL as follows (this constraint relies on the previously defined functions 'populations' and 'workloads' that return the sets of these elements in the associated models):

```
package Foundation::Core
 context Abstraction inv:
  self.stereotype→exists(name = "ACtoQN")
  implies
  self.populations→forAll(w : ModelElement |
   self.workloads→one(p : ModelElement |
    p.name = w.name and
    (w.stereotype→exists(name = "OpenWorkload") implies
    p.stereotype→exists(name = "OpenPopulation")) and
    (w.stereotype→exists(name = "ClosedWorkload") implies
    (p.stereotype→exists(name = "ClosedPopulation"))))
```

The queuing network model can be accessed via XMI [41] or JMI [59] metadata interfaces, and is in a form amenable to solution using the exact Mean-Value Analysis (MVA) algorithm [30] without further translation. We implemented the algorithm and applied it for various user population arrival rates to produce the results shown in Figure 3.6. The insights that such results provide are the key benefit of analysis. In this case the user experience seems relatively insensitive to load, whilst the editors can be significantly hindered. Resource utilisation figures show (perhaps unsurprisingly) that this is due to contention for the database, suggesting possible design modifications.

## 3.3 Analysing distributed-system architectures

Having presented our general approach to deriving analysis models from designs, we now discuss approaches to analysing software architectures. Architectural components present a challenge to the method, as they have complex behaviours that the developer should not be expected to model.

The use of transformations provides the means to address this problem. The architectural components of designs can be denoted using an appropriate

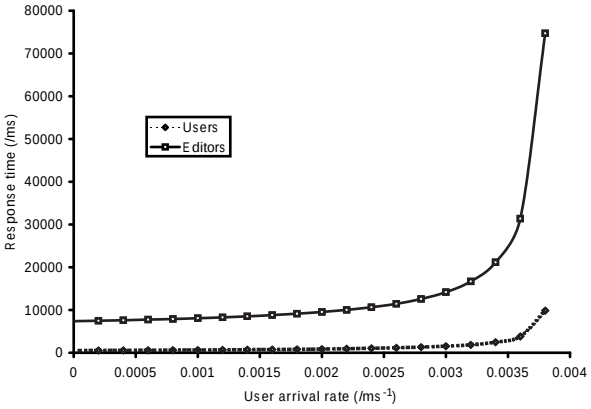


Figure 3.6: Response time results for varying user arrival rates

domain specific extension. The transformation can proceed from this view of the architecture, incorporating knowledge about the behaviour of the architectural components into the resulting analysis model.

In ongoing work we are investigating the performance of Enterprise JavaBeans (EJB) servers in an effort to produce mappings to good predictive models from EJB application designs. EJB designs in UML are enabled by the EJB Profile [20], an ongoing standardisation effort.

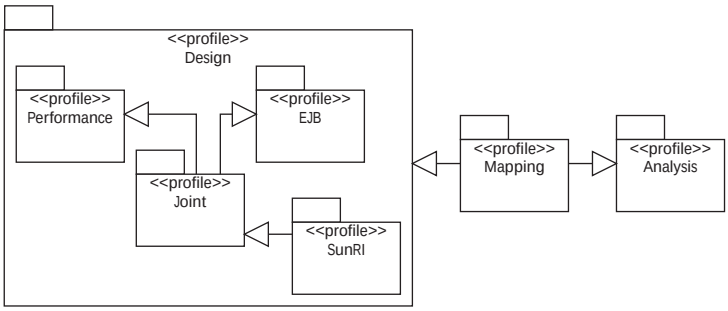


Figure 3.7: Profiles for the performance analysis of EJB designs

Figure 3.7 shows the profiles supporting the derivation of analysis models. We now use a combination of the Realtime profile and the EJB profile. These we combine by package subclassing. Because different implementions of the EJB container standard behave differently, we introduce an additional derived profile that includes a stereotype to distinguish a particular implementation, in this case the Sun Reference Implementation.

Figure 3.8 shows part of the design of an EJB auction-house application,

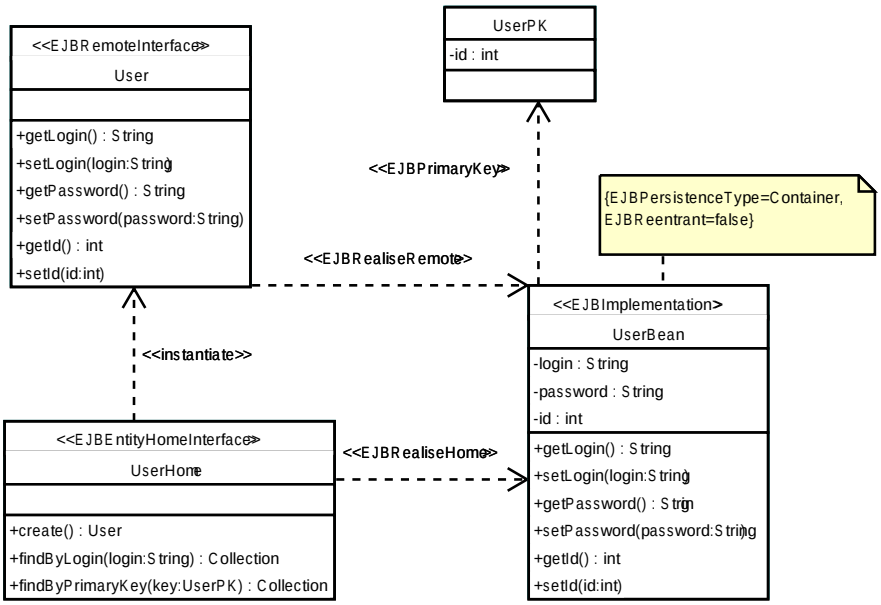


Figure 3.8: User classes in EJB auction application

including the User entity. The basic UML class diagram is enhanced with stereotypes from the EJB profile.

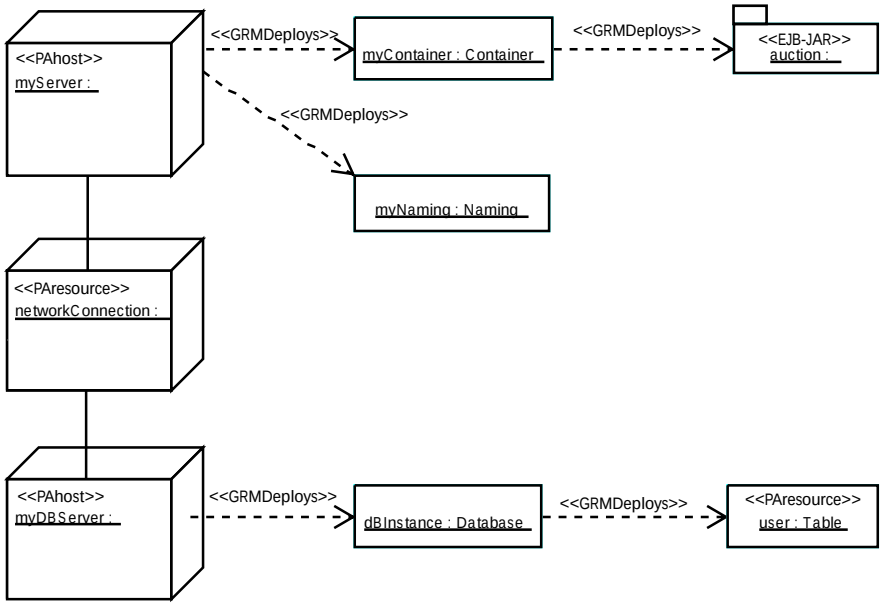


Figure 3.9: Deployment of EJB auction application

Figure 3.9 shows the deployment of the EJB application, including stereotypes from the EJB profile, the real-time profile (denoting deployment relationships and resource types), and a stereotype identifying the implementation of the EJB container.

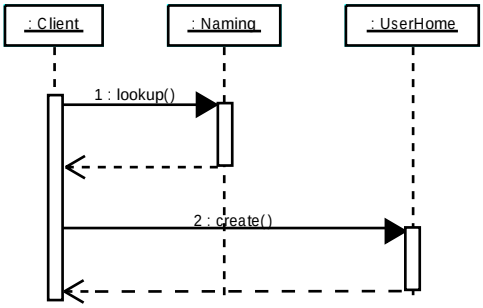


Figure 3.10: Sequence diagram for create-user usecase

The view of the behaviour of the application is still provided by a combination of use-case and sequence diagrams. However, the sequence diagrams necessarily omit the internal behaviour of the architectural components. For example, Figure 3.10 shows sequence diagram for the create user use-case as it appears to the applications developer. In fact, in order to create a new user, the EJB container must undertake a number of internal actions, including the creation of a proxy and instance objects, as shown in the more detailed sequence diagram in Figure 3.11. The mapping must take into account this extra behaviour when deriving the ultimate performance model. If necessary, two mappings may be used, the first to expand the design into an exploded version, including the actions of the architectural components, and the second to derive the performance model. This approach would have the advantage of providing a visible reference for the developer, effectively explaining how the predicted performance was derived.

Software and hardware architectures potentially provide a benefit to our method. Because their implementation changes relatively slowly, it is possible to incorporate default values for performance. These will not necessarily be absolutely accurate, but their relative values may be accurate. For example, the relative speeds of information exchange in memory, to backing store and over a network can be determined. Incorporating this knowledge into a mapping would allow a 'rule-of-thumb' assessment of application performance that would certainly have a good chance of identifying poor caching strategies.

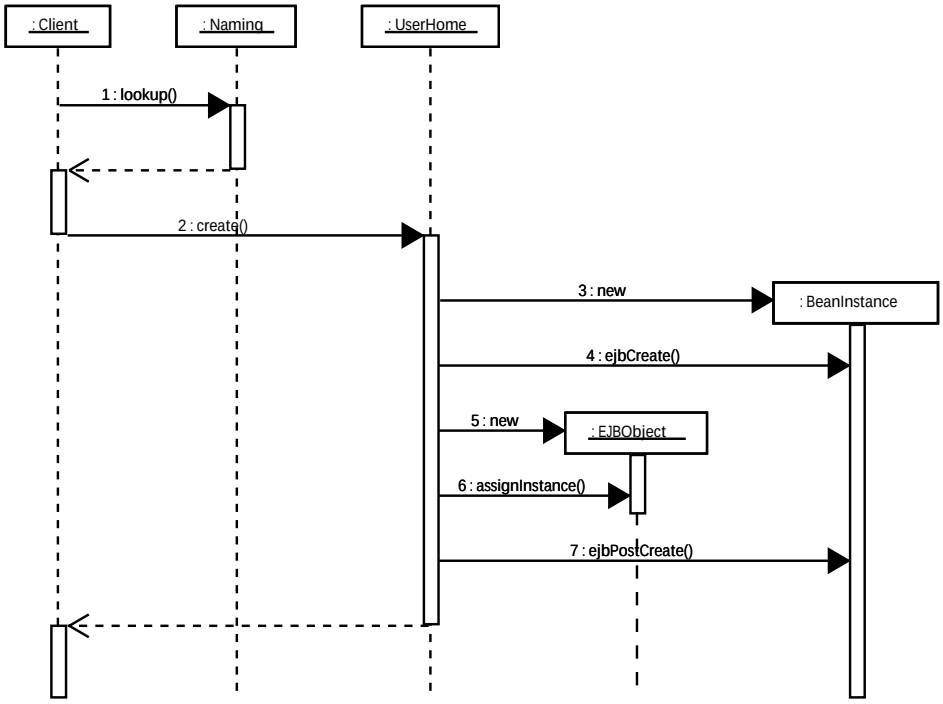


Figure 3.11: Exploded sequence diagram for create-user usecase

## 3.4 Other Non-functional Properties

Our work thus far has concentrated on deriving performance values. These are obviously of paramount importance to application services. However, just as vital are approaches to managing the reliability of services. The paper [52], included as supplementary material to this report, details our early work in this area.

## 3.5 Related Work

The notion of deriving mathematical models from UML diagrams is not a new one, and a complete survey is beyond the scope of this paper. Efforts can be broadly classified into those that seek to define a formal semantics for the language [58, 29, 22], those that derive models for functional properties [34, 21], and work such as our own that derives models for non-functional properties [2]. Previous work suffers in two respects:

Firstly, a formal semantic for the UML has not been standardised, and it is unlikely to be in the future. The implication for efforts to derive formal

models is that no assertion about the validity of the derivation can be made. Two methods can derive functional models that associate different behaviour with the same UML model, and there is no way to chose between them. The meaning of the UML model is unclear in isolation, with different analysts embedding different personal assumptions in their analyses. This is also true of the real-time profile, which defines its semantics in terms of an MOF-style domain model. Different performance models will produce different results, and the domain model can only be used to qualitatively assess the extent to which the derivation respects the domain semantics. Even if a completely formal semantic were available it is unlikely to be computationally feasible to determine the consistency of derivations. Formalisms often make simplifying assumptions to render analysis feasible (e.g. the memory-less property in performance models) that conflict with domain semantics, so some notion of approximate consistency is also required.

The second problem is that previous work adopts descriptions of derivations, such as axioms, graph-grammars or informal associations that are not standard to the MDA. Moreover, the formal models themselves are described in terms of their abstract mathematical structure, with no consideration given to their management in an engineering environment. Often the details of the transformation are captured by a research prototype that serves as a reference implementation, and provides the only practical support for the method. This fails to realise a key benefit of a standards-based approach, which is to enable a market in robust industrial tools that support it.

Our approach addresses both these issues by expressing both the derivation and all models using UML. The UML design becomes closely coupled to its assigned semantic in whatever formalism, because the mapping and the derived models are stored in the same tool or repository with it. Any standards-compliant tool is capable of storing these models.

We believe that our approach to capturing relationships between UML models and formal models is compatible with most of the very useful derivations hitherto proposed, and also with methodologies that explicitly prescribe formal modelling, for example [56] and [5]. Our example derives a queuing network model from the performance sub-profile of the real-time profile, and so resembles [48].

Our use of OCL constraints at the meta-model level is related to work to define inter-diagram consistency within the UML [26], particularly that of the NEPTUNE project [6] which developed the OCL Evaluator [9], although we are concerned with separate semantic domains and embed our constraints in

profiles, rather than directly into the meta-model. The concept of defining the semantic of a UML diagram in terms of another UML diagram is similar to the work of the Precise UML group [11], who include semantic domains and transformations in meta-models.

# Chapter 4

# SLAng Semantics

## 4.1 Introduction

The integration of SLA information with analysis and architectural models of systems requires a means to model SLAs and an understanding of what the models mean. In this chapter we formally define the meaning of the SLAng language. The resulting semantic model forms the basis for a UML extension allowing the modelling of SLAs in the same context as the analysis models and architectural designs discussed in previous chapters.

The meaning of SLAng is formally defined in terms of the behaviour of the services and clients involved in service usage. Benefits of the formal semantic include the reduction of ambiguity in the meaning of the language and the means to check the semantics to ensure the absence of inconsistencies and loopholes. The style of semantic definition used aims to be user-friendly, allowing it to serve as a reference for human negotiators. It also provides a formal basis for comparisons between SLAs, and an abstract reference model of systems employing SLAs that can guide implementation and analysis efforts. These latter facilities address two types of compositionality for SLAs: *inter-service composition* in which required QoS levels are compared to offered QoS levels, and *intra-service composition* in which the QoS levels offered by a service are related to the levels provided by its components.

We use the Unified Modelling Language (UML) to model the language, producing an abstract syntax. We embed this language model in an object-oriented model of services, service clients and their behaviour. The presence of SLAs, instances of the language model, constrains the behaviour of the associated services and service clients. The constraints are defined formally using the Object

Constraint Language (OCL), with accompanying natural language descriptions, and define the semantics of the language. The semantics are easily understood in the context of the service model. We exemplify the approach in Section 4.3 by describing SLAng's semantics for ASP SLAs.

Inter-service composition requires the matching of desired service levels, expressed as target SLA terms, with offered service levels. SLAs are deemed to be compatible if the offered SLA permits no behaviours (according to the service model) that would violate the target SLA. Inter-service composition is discussed in Section 4.4.

Intra-service composition requires a specification of the behaviour of a system extraneous to SLAs. We therefore employ our combined service and language model as the basis for an extension of UML. This allows the modelling of services and SLAs in the same context as more detailed design information. The semantic model of services and SLAs informs analysis activities that determine the emergent qualities of composed systems. It can also inform the development of SLA aware services, as it provides a reference model for the behaviour of such systems. The UML extension is provided in the form of a QoS catalogue appropriate to the proposed 'UML Profile Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms' (henceforth 'QoS profile'), and is described in Section 4.5.

The next section describes SLAng in more detail. Section 4.3 gives an example of the semantic definition of SLAng by presenting the semantics of ASP SLAs. Section 4.4 discusses the compatibility of SLAs. Section 4.5 presents the QoS catalogue for the ASP characteristics in SLAng. Section 4.6 discusses the differences between SLAng and other SLA languages in terms of approach and style of semantic definition.

## 4.2 SLAng

SLAng meets the need for an SLA language to support construction of distributed systems and applications with reliable QoS characteristics. The syntactic structure and semantics of SLAng are defined with reference to a model of distributed system architecture. The model defines the scope of the language, and has assisted in identifying the service usage scenarios and parameters that SLAng must represent. The reference model is shown in Figure 1.1[1].

In our model, applications are clients that use application services to deliver end-user services. Application services are services with an electronic interface,

[1]Slightly modified from [28]

such as web services or J2EE or .NET components. Containers host application services and are responsible for managing the underlying resource services for communication, transactions, security and so forth. Networks provide communication between services and storage can implement persistence for containers.

All SLAng SLAs include: An end-point description of the contractors (e.g., information on customer/provider location and facilities); contractual statements (e.g., start date and duration of the agreement); and Service Level Specifications (SLSs), i.e. the technical QoS description and the associated metrics.

SLAng defines six different types of SLA, corresponding to service usages present in the model. These are divided into Vertical SLAs, in which the service provides technical support for the client, and Horizontal SLAs, in which the client subcontracts part of the functionality of a service to a service of the same type. The hierarchical structure of SLAng's syntax subdivides the SLS terms into SLA-type specific groups. The terms are further subdivided into client, provider and mutual responsibility clauses.

The Vertical SLAs are *Hosting* (between service provider and host), *Persistence* (between a host and storage service provider) and *Communication* (between application or host and Internet service providers). The Horizontal SLAs are *ASP* (between an application or service and ASP), *Container* (between container providers) and *Networking* (between network providers).

The SLAng syntax is defined using XML Schema. The choice of XML as a basis for the language reflects the popularity of XML in the domain of distributed systems. In particular XML documents are frequently used to provide service meta-data [64, 63] and deployment descriptors [60]. By adopting XML as a basis for SLAng we seek to ease the integration of QoS adaption and negotiation technologies depending on SLAng statements with existing Internet service technologies.

## 4.3 SLAng Semantics

A formal semantic definition for SLAng has at least the following advantages:

1. Ambiguity concerning the meaning of the SLA is limited to disagreements concerning correspondence between semantic elements and the real world. Since the semantic elements are more specific than the corresponding language elements, ambiguity is reduced overall.

2. The implications of the semantic definition can be tested and refined by generating SLAs and assessing whether the semantics are reasonable.

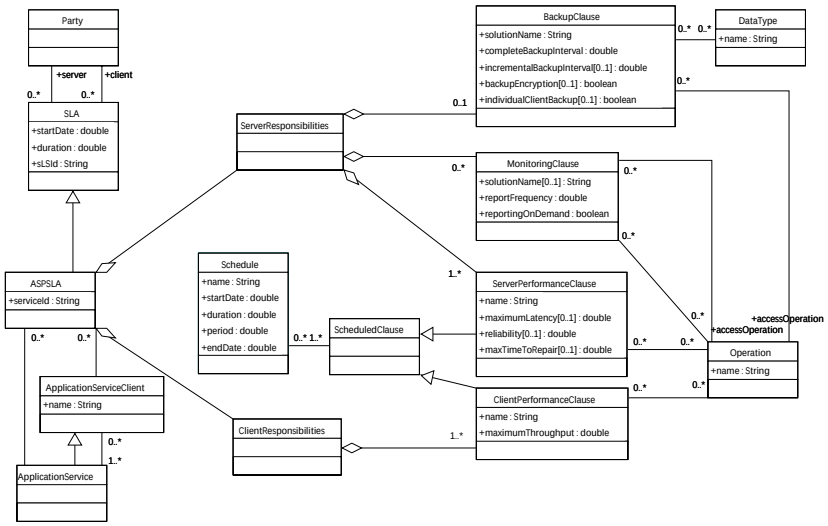


Figure 4.1: Abstract Syntax of SLAng

3. The semantic definition forms an objective basis for definitions of relationships between SLAs, in particular matching desired service levels to offered service levels as required for inter-service composition.

4. The semantic definition provides a reference for service implementations that must conform to SLAng SLAs.

### 4.3.1 Approach

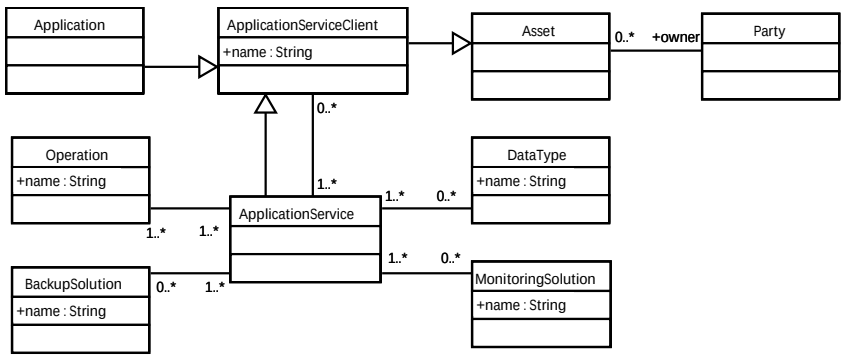


Figure 4.2: Refined reference model for application service provision

The approach taken to formalising SLAng adapts the approach of the Precise UML group to formalising UML [11]:

1. The language itself is modelled in UML. This creates a meta-model, or abstract syntax. Statements in the language can be regarded as instances of this model. UML is based on a defined abstract syntax, which serves as a point of attachment for semantic descriptions, meta-data interfaces and notations. In the case of SLAng, the XML schema provides the primary definition of the language. It was therefore necessary to manually translate this schema into UML. Figure 4.1 shows the abstract syntax for ASP SLAs. It reflects the hierarchical structure of the XML schema for SLAng, with the top level defining the type of the SLA, and separate clauses for provider and client responsibilities (there are no mutual responsibilities in this example).

2. The parties and services involved in the agreement are modelled. In the case of SLAng, the informal reference model provided a starting point. This was translated into UML and refined. Figure 4.2 shows the refined reference model for application service provision. Additional concepts are introduced to support the definition of semantics for terms in SLAng. For example, an ASP SLA can specify the types of backup and monitoring solution used. BackupSolution and MonitoringSolution are introduced to allow the assertion that the solutions used in the real world must correspond to those specified in the SLA.

   Defining the type of software used for backup and monitoring may seem to contribute little to the principle goal of ensuring QoS for distributed applications. However, the industrial requirements on which SLAng is based indicate that clients frequently required this. The capacity to formalise clearly such apparently informal constraints is a significant benefit of the approach taken.

3. The behaviour of the parties and services involved in the agreement is modelled, using the reference model as a basis. Figure 4.3 shows the behaviour of application services and their clients. The primary interaction in this case is the ServiceUsage, an interval of time during which the client is invoking an operation of the service. Operations are abstract capabilities of the service that can be invoked by the client. The client must be able to detect the beginning of the usage and its successful completion. Also, if the usage appears to have completed, the client can detect whether a failure occurred.

4. The language model is related to the elements that the SLAs are intended to constrain. This can be seen in Figure 4.1, where the classes Appli-

cationService, Operation, ApplicationServiceClient, BackupSolution and MonitoringSolution are reference model elements, associated with the relevant clauses in the language model.

5. The semantics of SLAng are defined by the constraints imposed on the behavioural model by the presence of SLA elements. These are expressed using OCL constraints defined in the context of the SLA clauses. The SLA is associated with a service and service client, so the constraints can refer to these entities and place conditions upon them. For readability, the constraints are also expressed in natural language.

The complete set of constraints defining the meaning of the ASP SLA are documented in [55].

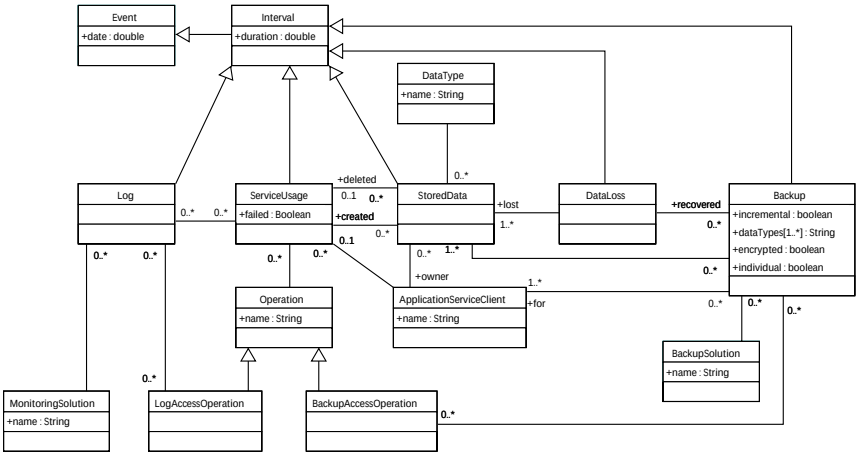


Figure 4.3: Behavioural model for application service provision

### 4.3.2 Example

We now present the OCL definition of the constraint used to define the meaning of maximum latency and reliability.

Reliability in the ASP SLA is defined in terms of failed or overdue usages of the system. Each failure gives the client leave to assume an interval of service outage equivalent to the inter-invocation time if they were using the service at the maximum allowable rate. This is because if the client does not use the service then they cannot reasonably claim that it is unavailable, and we do not assume that the provider can be trusted to report their own outages honestly

(neither can the client necessarily be trusted – a trusted third party could poll the service instead, with the same definition of reliability being required). The maximum allowable rate is the strictest client performance clause applying at the moment the failed operation is invoked.

Server and client performance clauses in the SLA are associated with schedules. A schedule defines when the clause applies, by specifying a start date, an end date, a duration and a period. The clause applies repeatedly, lasting for the duration, then becoming inactive until the period is complete. Multiple schedules can be associated with a clause to allow the specification of complex timing, with the interpretation that the clause applies when any one of its schedules apply. For example, five schedules with durations of 8 hours, periods of 1 week and start dates offset by 1 day can be combined to specify the composite schedule 'every working day'.

The following OCL definitions rely on the models presented in Figures 4.1, 4.2 and 4.3. Each is defined in the context of the class ServerPerformanceClause in Figure 4.1, meaning that the constraints apply to all instances of that class, i.e. all server performance clauses in the real world. The operations and usages associated with the clause are referred to in the constraints by navigating across the associations present in the UML models, using the dot operator (.) and the name of the opposite association end, usually the name of the associated class with a lower-case first character. Attribute values and OCL operations are referred to using the same syntax. OCL operations are side-effect free operations defined in the context of classes, and we have used them to decompose complex constraints. By convention we have omitted their signatures from the operations compartment of our diagrams.

Reliability constraint: Proportion of downtime observed to total time that the clause applies must not be greater than the percentage of permitted failures (1 - the reliability).

**context** ServerPerformanceClause **inv**:
self.operation→forAll(o |
totalDowntime(o) < (applicationTime ⋆ (1 - reliability)))

The following additional OCL operations support the definition of the reliability constraint:

**context** ServerPerformanceClause **def**:
−− *Returns the client performance clauses governing the performance of operation o*

```
at time t.
 let applicableClientClauses(t : double, o : Operation) =
  sla.clientResponsibilities.clientPerformanceClause→select(c |
c.schedule→exists(s | s.applies(t)))

-- An expression for the maximum throughput with which the client can use an op-
eration at time t, or −1 if there is no limit
 let minThroughput(t : double, o : Operation) =
  if applicableClientClauses→isEmpty() then −1
  else applicableClientClauses→iterate(
   c : ClientPerformanceClause, minTP : double |
    minTP.min(c.maxThroughput))

-- Amount of downtime observed for a failure at time t. This is 1 / the most restric-
tive throughput constraint applicable at the time
 let downtime(t : double, o : Operation) : double =
  if minThroughput(t, o) <= 0 then 0
  else 1 / minThroughput(t, o)

-- Total amount of downtime observed for the operation
 let totalDowntime(o : Operation) : double =
  o.serviceUsage→select(u |
   (u.failed or u.duration > maximumLatency) and
   schedule→exists(s | s.applies(u.date))
  )→collect(u | downtime(u.date, o))→iterate(
   p : double, sumP : double | sumP + p)
```

The constraint also relies on the definition of the following operations: *applicationTime*, defined in the context of ScheduledClause, the superclass of ServerPerformanceClause, which evaluates to the total time for which a ScheduledClause is applicable; and *applies*, defined in the context of Schedule, which evaluates to true if the schedule applies at the specified time and date.

The complete set of operations, combined with the constraint, defines the effect of the SLA on the environment. The proportion of failed or overdue service usages may not exceed 1 − the specified reliability. The definition is unambiguous and consistent, providing a strong reference for parties employing SLAs. The semantics also support activities related to service composition, as discussed in the subsequent sections.

## 4.4 Inter-service Composition of SLAs

We say that an SLA B is *compatible* with another A if the set of allowable behaviours for B is a subset of those for A. In other words, a system conforming to B would never violate A, and would hence be perfectly acceptable to a client requiring A.

The notion of compatibility supports inter-service composition. One service can require another and express its requirements using an SLA. Any service that both provides the required functionality and offers a compatible SLA can be composed to fulfil the requirements.

Our definition only allows the comparison of fully specified SLAs. SLAs include restrictions on client behaviours and it might seem preferable to allow the comparisons of SLAs with requirements relating only to server behaviour. However, this would be dangerous due to interactions between SLA terms. For example, in the definition of reliability presented in the previous section there is a relationship between the invocation rate constraint on the client and the reliability. Therefore, an SLA offering higher reliability and a faster rate is still not necessarily compatible with an SLA with a slower rate, as the absolute number of failures may be the property of concern for the client, rather than the absolute number of successes.

A possible generalisation of the notion of compatibility would be the ability to compare an SLA C against another D where sets of values or ranges were specified for each parameter in D, with the interpretation that C is compatible if it is also compatible with some specific valuation of D within the ranges.

One possible procedure for checking if B is compatible with A is to employ the semantic model as a meta-model. A could then be associated with the service model and the constraints of A checked for the set of all behavioural models acceptable to B (which might be thought of as the set of all traces of system behaviours). Clearly this approach is potentially extremely computationally expensive, and unworkable if the possible behaviours of B are infinite. It would be preferable to employ a theorem prover to establish the validity of $B \rightarrow A$ where $B$ is the union of the constraints in B and $A$ the equivalent for A. Future work will investigate this approach.

Our definition of compatibility is similar to that of *conformance*, defined in relation to the language QML [14], in which a contract A conforms to another B if its constraints are *stronger*. Each SLA dimension defined for a contract in QML has a direction indicating an ordering over values of the metric, with higher values implying stronger constraints. Conformance of contracts can therefore be assessed by comparing the values of corresponding dimensions in two contracts.

In comparison our definition of compatibility is hard to check and somewhat inflexible. However, its basis in the semantic definition of the SLA terms, rather than on user-defined ordering of metric spaces suggests that the concept offers safer guarantees that requirements will be met, particularly in the presence of dependencies between SLA terms as discussed above.

## 4.5 QoS Catalogue for SLAng

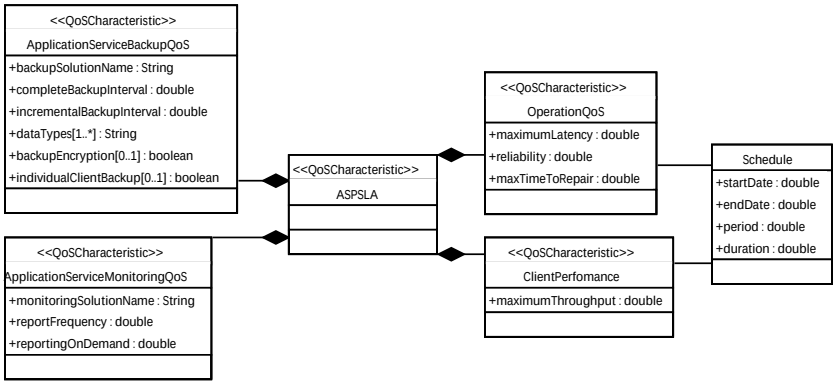


Figure 4.4: ASP QoS Characteristics for SLAng catalogue

Reasoning about internal composition of SLAs is a special case of QoS prediction, in which some components are governed by SLAs and the system as a whole will conform to, or offer an SLA. QoS prediction requires a view of the system in which the effect of the components on the quality attributes are known. It may also require sophisticated analysis to determine the emergent QoS values. UML potentially offers such a view. It can represent the logical structure, deployment and behaviour of a service. It can also be extended using profiles to enable the description of QoS properties. In this section we describe how SLAng SLAs may be modelled using UML, and how their semantics enables analysis and implementation activities.

A profile is a semantic extension for UML allowing it to naturally model domains of interest [44]. It is common to define the semantics of a profile with reference to a domain model [13]. The interpretation is that elements in the UML model labelled with stereotypes correspond to instances of the domain model. The Object Management Group (OMG) has standardised profiles for particular application areas. The use of standard profiles ensures reusability for models, and interoperability for tools that operate on annotated model data.

The QoS profile [36] is currently a proposed standard. It allows the modelling of QoS characteristics as classes, and QoS values as instances of these classes. QoS values can be associated with other model elements to indicate behaviour or requirements. The proposal compensates by providing a catalogue of QoS characteristics with informally defined semantics.

Rather than define a new profile to represent services and SLAs we have reused the QoS profile by defining a QoS catalogue for SLAng. Figure 4.4 shows the SLAng catalogue for ASP SLAs. The SLA terms are defined as QoS characteristics and therefore inherit the definitions provided by the semantic model, analogously to defining a profile directly according to a domain model.

The QoS profile allows corresponding QoS values to be attached to messages in a UML 2 communication diagram using one of three stereotypes: QoSContract, QoSRequired and QoSOffered. In all cases we state that the recipient of the message corresponds to the service associated with the SLA in the semantic model, and the sender corresponds to the service client. These elements are assumed to behave in accordance with the SLA terms. Where QoSOffered is defined together with QoSRequired or QoSContract there is the opportunity for a tool to check the compatibility of SLAs according to the compatibility criteria defined in the previous section.

Figure 4.5 shows an example model including a SLAng ASP contract governing the interaction between a client and an online auction service. Constraints on the bidding operation are shown.

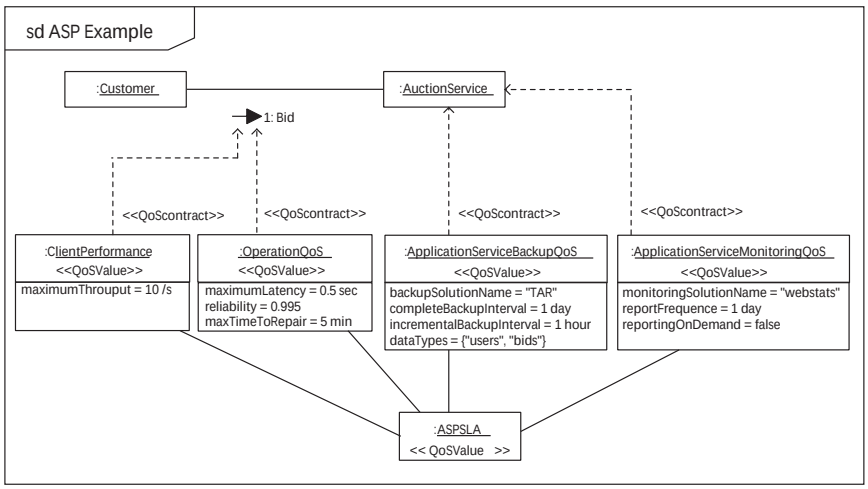


Figure 4.5: Example communication diagram including SLAng QoS values

The ability to represent SLAs in UML is a necessary but insufficient condi-

tion to enable reasoning about QoS properties. It is also necessary to represent the impact of system components not directly governed by SLAs, and in some cases to support an analysis method for determining the emergent characteristics of the system. The 'Profile for Schedulability, Performance and Real-Time Specification' (henceforth 'real-time profile') [42] provides these facilities for systems in which performance is an issue and resource utilisation and scheduling have the greatest performance impact. It supports the derivation of models such as queuing networks or Petri nets. Other efforts are underway to extend UML to describe reliability properties [52].

[36] shows how QoS characteristics can be defined using the domain model of the real-time profile, allowing direct analysis. Because we use an alternative semantic model this approach is not directly available to us. However, it is perfectly possible to use our QoS characteristics simultaneously with real-time profile annotations with the interpretation that the real-time measures are an approximation of the service levels. We make this correspondence explicit by introducing a new stereotype `<<approximation>>` defined in its own profile 'ValueRelationships'. This stereotype is applicable to dependencies between tagged values, and its interpretation is that the dependent is an approximation to the target. In future work we will consider how this relationship can be formalised, and what its implications are for design consistency.

We have now provided adequate notation to present a complete picture of the behaviour of an ASP system in terms of performance, including SLA terms and with sufficient detailed to permit analysis.

UML is the design language of choice when adopting a Model Driven Architecture (MDA) [13] development strategy. Model transformation is a key technology in the MDA, used to deploy business knowledge automatically in new technical domains. Transformations rely on the semantics of both source and target domains. By providing a rich semantic and representation for SLA-aware systems our QoS catalogue potentially serves as a starting point for implementations of such systems, according to MDA principles and benefiting from model transformations. This approach is described in more detail in Chapter 5. In Chapter 3 we show how MDA techniques can be used to enable automatic analysis, giving an example in which performance analysis proceeds from designs expressed using the real-time profile. The SLAng semantics therefore provides a reference for both analysis and implementation efforts relating to internal composition of SLAs.

## 4.6 Related Work

Our approach to defining the semantics of SLAng is derived from the work of the Precise UML group, who define an abstract syntax for UML and lend it semantics by associating a semantic domain. Our contribution to the approach is to demonstrate its application at a high level of abstraction, and including sophisticated quantitative properties. In this section we compare the semantic definition of SLAng to that offered by other SLA languages. We also briefly describe features of other languages lacking in SLAng, and their potential impact on the semantic definition.

The QuA project adopts the most rigorous approach to defining the semantics of QoS properties [57], although to our knowledge they have yet to define a concrete syntax for representing SLAs. According to their model, all QoS properties are related to the performance of a service, which supplies a set of operations. Input and output messages are causally related by operation invocations. Output messages are characterised by a set of variables. A set of error functions are defined over the difference vector between an observed output trace for a particular input trace, and the ideal trace as it would be observed were the service deployed on infinitely fast equipment operating without error. SLAs are defined using constraints on the values of error functions.

It is possible to see correspondences between our semantics and the QuA approach. In our case the service model defines the information available concerning service operation, and the OCL constraints provide a concrete representation of the error functions. Features of our semantics not obvious in the QuA approach are constraints independent of a service model assumption, such as the constraint that the service provider must be capable of providing the monitoring solution specified in an ASP SLA, and the ability to constrain client behaviour (in QuA terms, the input trace).

QML[14]defines a type system for SLAs, allowing the user to define their own dimension types. Whilst this makes the language highly extensible the meaning of the individual metrics in the context of the system is not formally established. Since exchange of SLAs between parties requires a common understanding of such metrics, this can be regarded as a serious deficit. QML does define a rigorous semantic for both its type system and its notion of SLA conformance however.

QML and WSOL [62] both provide type systems for SLAs, allowing the same SLA to be described in abstract and also instantiated with specific values. This provides more guidance to developers of new SLAs than our use of XML schemas. Because SLAs require common understanding of terms between par-

ties, it is to be hoped that new types of SLA will be defined only infrequently. However, the generalisation relationships between SLAs are potentially helpful in structuring a family of SLA types. WSOL provides additional reuse facilities [46], including template instantiation and reuse of definitions.

WSOL [61]and UniFrame [7]SLA specifications both rely on the specification of measurements in external ontologies. These ontologies are structured natural language descriptions of measurements, including advice on how they should be taken and their interdependencies.

WSLA [18] is another XML based web services SLA language. All measurements are assumed to be provided by a web service encapsulating monitors. No constraints are placed on the implementation of such monitors, so a common understanding of their role remains external to the definition of the language.

WSLA provides the ability to create new metrics defined as functions over existing metrics. This is useful to formalise requirements expressed in terms of multiple QoS characteristics, without impacting on notions of compatibility of SLAs. The semantic for expressions over metrics is not formally defined, but no barrier prevents such a definition.

WSLA is an XML language, structured in such a way that monitoring clauses can be separated from contractual terms for distribution to a third party. We are interested in supporting third party monitoring schemes with SLAng. Precisely how such schemes will work will require careful modelling to inform the design of the metrics. However, this principle of syntactic separation is clearly useful.

WSOL provides an additional syntax to interrelate service offerings. Relationships indicate substitutability of SLAs in the event of a violation. Such facilities are clearly useful for a language

WSOL and WSLA allow the definition of management information, including financial terms associated with SLAs. These are not presented with a defined semantic (although WSOL claims the need for financial ontologies), but are clearly a desirable feature of SLA languages lacking in SLAng. Both languages also define management actions, including notifications in the event of SLA violations.

The CORBA Trading Object Service [37] allows the advertisement and selection of services offers based on constraints over typed properties. These properties can include QoS specifications, and generally can take any IDL type. Their semantics is not formally defined; neither are external ontologies specified. It is therefore up to the trader and its clients to agree an interpretation for the properties.

QuO [1] is a CORBA specific framework for QoS adaption based on prox-

ies. It includes a quality description language used for describing QoS states, adaptations and notifications. Properties in the language are defined to be the result of invoking instrumentation methods on remote objects. Like WSLA, no formal constraints are placed on the implementation of these methods.

# Chapter 5

# Modelling Electronic Services

This chapter presents a profile for modelling systems of electronic services using UML. For the purposes of this chapter we consider a restricted definition of electronic service aimed at business systems, distinct from the more general notion of application service used in the rest of this document. Electronic services encapsulate business services, an organisational unit focused on delivering benefit to a consumer, to enhance communication, coordination and information management. Electronic services are application services for the purposes of communication, but they are also associated with behavioural specifications in the form of workflow descriptions, and with business concepts such as provisioning and resource management. The intent of electronic services is to enable organisations to structure their businesses in order to gain the benefit of workflow management, enterprise resource planning, process modelling and electronic commerce.

The relevance of Electronic Service Systems (ESSs) to the TAPAS project is two-fold. Firstly, TAPAS is concerned with service provision, and electronic services provide a rich example of a service model. The electronic service model includes business and resource views that will be relevant to organisations seeking to deploy trusted and QoS aware applications. Secondly, the approach taken to modelling electronic services that we outline in this chapter could equally well be applied to the TAPAS project. The benefits of the approach are to document ESSs, to provide platform independent models of ESSs as a precursor to MDA development processes, and to enable reasoning about the behaviour of composed ESSs.

In this chapter we define a UML profile for modelling ESSs. The profile is associated with a domain model that describes its semantics. As discussed in Chapter 2, this is a common approach for defining the semantics of a UML extension. We also provide an operational semantics for electronic services, formalising our notions of service behaviour and composition.

For the approach to be applicable to TAPAS we need to determine both a business model for the composition of trusted and QoS-aware services and an architectural view of the TAPAS platform. We envisage the latter emerging from a combination of the SLAng semantics (Chapter 4), which describe the behaviour of QoS aware applications, and the TAPAS platform specification [12], which will determine the architecture of such systems. A unified model of TAPAS architecture and business processes will hopefully be the focus of future consultation with our project partners.

The remainder of this chapter is structured as follows: Section 5.1 introduces ESSs in more detail; Section 5.2 describes the semantic domain model for ESSs including the operational semantics; Section 5.3 describes the derived profile; Section 5.4 gives an example of a service model based on the freight industry.

## 5.1 Electronic Service Systems

An electronic service is a set of metadata, communication interfaces, software and hardware supporting a business service [33]. A business service is a bundle of coordinated business capabilities (the content of the service) associated with provisioning mechanisms that establish the conditions under which clients, whether external or internal to the business, can access the capabilities of the service.

Business services encapsulated by electronic services benefit from additional communication and provisioning channels, but further, they permit the automated coordination of capabilities, resources and information, both within and between organisations. This gives rise to Electronic Service Systems (ESSs), in which the services are integrated using auxiliary components such as workflow engines for coordination, databases to store knowledge about the state of the enterprise, and Electronic Service Management Systems (ESMSs). We characterise ESMSs here as combining the various capabilities of databases and workflow engines to provide viewpoints and control of the enterprise to management, citing experience of the HP Service Composer and the DySCo (Dynamic Service Composer) research prototype.

The notion of a 'business service' enables the management within an enter-

prise of 'capabilities' to deliver some benefit to a consumer. The term 'capability' refers to the coordination of simpler tasks to achieve an end; the concept is used to raise the level of abstraction when describing the way that a business behaves. When describing business services, capabilities are divided into those involved in 'provisioning' the service, and those providing the 'content' of the service. The content of a service is the set of capabilities that deliver the benefit of the service to the client. For example, the content of a freight service refers to the capability of moving goods from one place to the other. Provisioning refers to the business channel [15] between the provider and the consumer of a service. In the example, provision covers selection, product offer, pricing, and interaction processes that the freight company applies to its customers. Content and provisioning are complementary aspects of a service. On the one side, the provisioning logic depends on the capabilities that the provider can support. On the other side, the capabilities made available to consumers depend on the provisioning logic adopted by the provider. In the example, the option of delivery tracking might be made available only to selected customers. The example is based on previous research in the freight domain [31].

Because business services require communication between the provider and the consumer it is natural to provide interfaces to business services using communication technologies such as computer networks, and the software that supports this such as middleware for distributed systems.

Middleware services and computing resources also provide the opportunity to implement new business services with a highly automated content, and this is an expected benefit of the electronic service model. However, despite the similarities, our notion of electronic services should not be confused with middleware services. Services must also be coordinated: Internally, to marshal the involved capabilities and resources and establish the relationship between content and provisioning; and externally, to manage the interaction between the service and its clients and environment. This coordination requires a view of the behaviour of a service. We therefore introduce an operational semantic for capabilities, presented in Section 4.3. This semantic is broadly compatible with workflow languages, suggesting that services could be both coordinated and enacted by workflow engines.

Our semantic also describes abstractly the effect that activities have on the information in their environment, for example the known locations of vehicles, or statistics such as the total revenue for a service. Such information can have a role in coordinating capabilities, and may be maintained and leveraged using databases or other accounting mechanisms.

There is also a need to manage the resources required by a service, which may be the role of an Enterprise Resource Planning (ERP) application. Generally, if electronic services are in place there will be the need and opportunity to integrate them using a technical infrastructure. We introduce the notion of an Electronic Service Management System (ESMS), informally defined as an application that includes coordination, information and resource management capabilities, providing business-oriented viewpoints and control over the services that it manages.

IT technology trends and the service model for business provide the context for electronic services. An enterprise adopting an electronic service strategy would structure its business as services, provide interfaces to those services using middleware technologies, coordinate and automate the services from a workflow-oriented perspective and implement a technological infrastructure to take advantage of the coordination and communication opportunities that are the key benefit of the electronic service model.

## 5.2 The ESS meta-model

The ESS meta-model is divided into two packages as shown in Figure 5.1. These partition the elements pertaining to services from those which represent management applications. The management component metamodel naturally depends on concepts from the service metamodel. The following sections present these metamodels in detail.

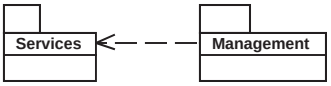


Figure 5.1: Subpackages within the ESS meta-model

### 5.2.1 The service meta-model

Figure 5.2 shows the part of the services meta-model related to the composition of capabilities into services. The elements shown are now described:

**Service** An electronic service as described in Section 5.1. Services have any number of provisioning capabilities, and a single top-level content capability (the capability to deliver the service). Services can be composed of sub-services, in which case the content capability coordinates the content of each sub-service, and each sub-service must have a provisioning

capability that makes a service offer to a role in the coordinating content capability.

**Capability** A behaviour which realises some benefit to the business, described by a workflow. The behaviour of capabilities is described formally in Section 4.3. Informally, a number of roles perform actions and cooperate to complete some task. Capabilities can be composed in a hierarchy. The workflow of a coordinating capability constrains the order of tasks in the component capabilities.

**CapabilityRole** A capability role identifies the behaviour of a worker or resource in a coordinated task. Capability roles can be assigned to actual business entities as discussed below.

**InformationItem** An identifier for a piece of information about an enterprise that is relevant to a task. Some workflow actions require information as a prerequisite and produce or process information as by-product of their enactment.

**Observation** Observations infer new information from existing information. This captures the idea that not all derived information is produced by a particular action. When the condition of the observation is satisfied then new information may be introduced by the observation expression.

Constraints defined over the meta-model further reinforce these informal semantics. For example, capabilities may not coordinate themselves. Constraints are expressed formally using OCL [44]:

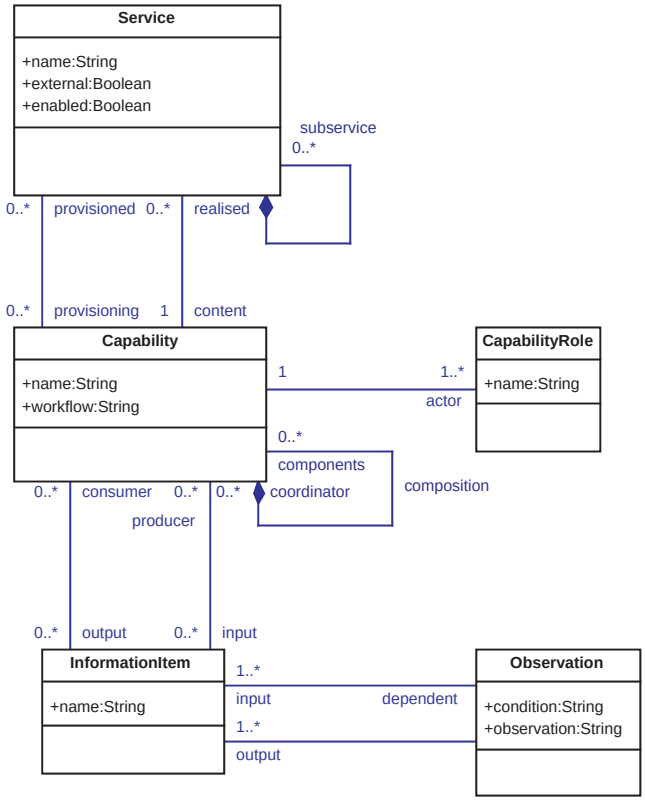


Figure 5.2: Capabilities view of the services meta-model

**context** Capability
  **def**:
    **let** allCoordinators = self.coordinator→union(
      self.coordinator→collect(c | c.allCoordinators))
  **inv**:
    **not** self.allCoordinators→exists(c | c = self)

Complementary to the abstract view of services are models of the business assets in an enterprise, and their assignment to capability roles to realise a service. Figure 5.3 shows the meta-model classes supporting such models.

**BusinessEntity** A business entity is a person, resource or system that can fulfil one or more roles in a capability.

**ServiceOffer** A service offer is made to a capability role (typically that of the 'customer'). That capability role must be associated with one of the provisioning mechanisms of the service.

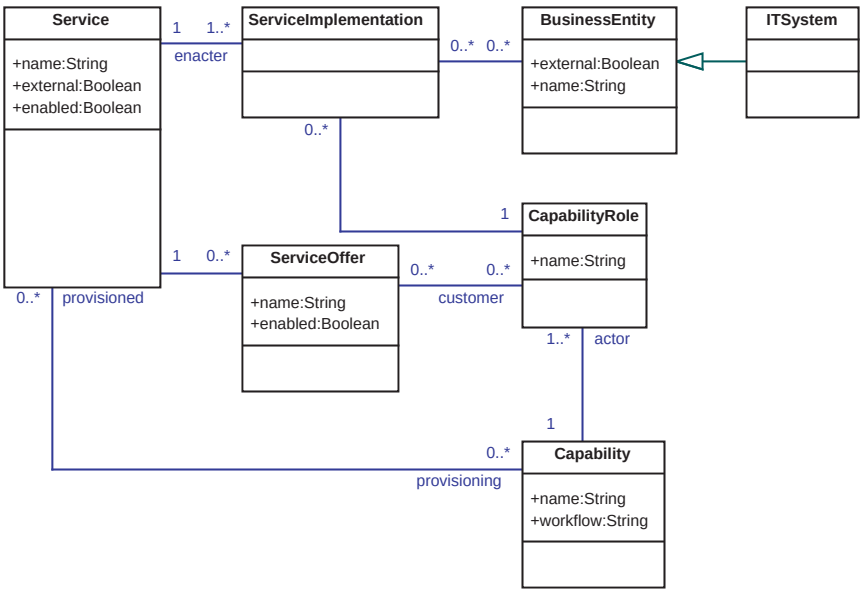


Figure 5.3: Implementation view of the services meta-model

**ServiceImplementation** ServiceImplementation captures the idea that business assets can be assigned to capability roles in order to make a service concrete. There is no explicit notion of service instance. However, if necessary business assets can be grouped to show those relevant to a particular scenario.

**ITSystem** An IT system is a computing system that can perform a role in a capability. Electronic services are intended to provide integration and automated coordination. This class allows the identification of the components providing these services, possibly as a prelude to an MDA-style development activity. Section 5.2.3 provides refinements of this stereotype to identify likely management applications.

Additional classes not shown in Figures 5.2 and 5.3 are now discussed:

**Property and HasProperties** Properties capture different types of metadata about capabilities. Such meta-information mainly refers to functional and non-functional requirements for a capability. For example, a property for a negotiation capability is to be usable only with a certain type of customers. The following classes inherit from HasProperties to enable the attachment of properties: BusinessEntity, CapabilityRole, Capability and Service. The properties mechanism maps onto the tagged-value mecha-

nism in UML in the profile definition.

**Group and Groupable** Experience with the HP Service Composer revealed the benefit of composing capabilities into loosely-grouped higher-level aggregates called 'clusters', in which capabilities exhibited functional overlaps, dependencies, mutual ownership or other subjective similarities. There is also often the need to group services into related offerings or 'service packs'. Finally, as stated above, a grouping mechanism addresses the lack of a concept of service instance by allowing the association of business entities that actually cooperate (since more than one entity can enact a given service role). Group and Groupable provide a single mechanism for hierarchical grouping. The following elements inherit from Groupable, and hence may appear in a Group: CapabilityRole, Capability, BusinessEntity, InformationItem, Service and Group. Grouping is implemented by UML's package mechanism in the profile definition.

### 5.2.2 Formal semantics for the service meta-model

In this section we formalise notions of information and coordination for capabilities, using the Structured Operational Semantics (SOS) style of [50], in which inference rules define the structure of a Labelled Transition System (LTS) intentionally. This definition contributes to the semantics of the profile by emphasising the definition of capabilities as coordinated activities whose behaviour is known, and by providing a high level constraint on workflow descriptions taken as values for the meta-attribute Capability.workflow. Our formalism is defined independently of specific workflow languages by omitting base cases for our rules. Instead, we assume that the workflow language employed allows us to make assertions such as:

$$\langle \Sigma \cup I, c \rangle \overset{\alpha: I \to O}{\longrightarrow} \langle \Sigma \cup O, c' \rangle \tag{5.1}$$

Meaning that a specific, isolated capability, $c$, in a system where the current information is represented by $\Sigma \cup I$, evolves to $c'$ after undertaking an action, $\alpha$, which causes some change, reflected by the transformation of the information $I$ to new information $O$.

Capabilities may evolve independently of each other, when not coordinated:

$$\frac{\langle \Sigma, c_i \rangle \xrightarrow{\alpha} \langle \Sigma', c_i' \rangle}{\langle \Sigma, \{c_1 \dots c_i \dots c_k\} \rangle \xrightarrow{\alpha} \langle \Sigma', \{c_1 \dots c_i' \dots c_k\} \rangle} \tag{5.2}$$

Even when coordinated, capabilities may perform uncoordinated actions ($A(c)$ yields the set of actions that a process $c$ can undertake):

$$\frac{\langle \Sigma, c_i \rangle \xrightarrow{\alpha} \langle \Sigma', c_i' \rangle \qquad \alpha \notin A(c_c)}{\langle \Sigma, c_c[\{c_1 \ldots c_i \ldots c_k\}] \rangle \xrightarrow{\alpha} \langle \Sigma', c_c[\{c_1 \ldots c_i' \ldots c_k\}] \rangle} \tag{5.3}$$

Coordinated actions may occur only when the coordinating process permits, and when all capabilities that can perform them are ready to do so simultaneously:

$$\frac{\langle \Sigma, c_c \rangle \xrightarrow{\alpha} \langle \Sigma', c_c' \rangle \quad \langle \Sigma, c_1 \rangle \xrightarrow{\alpha} \langle \Sigma', c_1' \rangle \ldots \langle \Sigma, c_i \rangle \xrightarrow{\alpha} \langle \Sigma', c_i' \rangle \quad \alpha \notin \bigcup_{c_f \in F} A(c_f)}{\langle \Sigma, c_c[\{c_1 \ldots c_i\} \cup F] \rangle \xrightarrow{\alpha} \langle \Sigma', c_c'[\{c_1' \ldots c_i'\} \cup F] \rangle} \tag{5.4}$$

A capability may have multiple coordinators in the metamodel. The interpretation of this is that the capability is a subcapability of its coordinator. It is therefore replicated for each coordinator. Shared sub-capabilities are not synchronized.

Note that an action may require certain information to be present and satisfy some condition before the action can be performed. Hence, coordination by shared memory is also possible for capabilities. Under the electronic service model, provisioning and content capabilities are not explicitly coordinated, hence this mechanism links these capabilities for a service. The provisioning capabilities create conditions under which the content capabilities are enabled.

Information in the system may arise naturally from the occurrence of actions. However, the progress of the system may depend on broader observations than those made in the context of a particular action. Hence we enable the modelling of observations that derive new information from that already present in the system:

$$\frac{\langle \Sigma \cup I, \Gamma \rangle \wedge \exists o : I \to O \in \Omega}{\langle \Sigma \cup I \cup O, \Gamma \rangle} \tag{5.5}$$

We do not prescribe the language used to specify observations. OCL would be a good candidate. The information prerequisites for the observation could be captured by a boolean expression, and then **let**-clauses could introduce new information. Note that it is possible to specify observations that lead to inconsistencies in the system information. Modellers should try to avoid this. One strategy for dealing with this is to rule that if multiple values can be derived

for an information item then the value of the information item is not known. However, in systems where action is preferable to inaction, this may not be safe.

For the purposes of assigning work the underlying workflow language must also associate actions with roles, although this association is not required in this discussion of coordination, as we assume that coordination is independent of the entities that implement roles. That is, an entity will eventually be capable of enacting all actions required of it during the evolution of the system.

The benefit of a formal semantic based on an LTS are in terms of simulation and analysis. A tool such as LTSA [32] can provide scenario-based validation of models. This can be used to assert safety conditions, fairness and liveness conditions, and to ensure the absence of deadlocks (presumably arising from capabilities failing to establish adequate preconditions for their successors). The use of information for coordination complicates such models, and can increase their state-space beyond feasibility. However, reasonable abstractions can usually be found.

### 5.2.3 The management meta-model

The management meta-model shown in Figure 5.4 allows the identification of common management components and their relationship to electronic services. We have not included modelling functional or structural relationships between management components as this is out of scope of our discussion of electronic services. However, such modelling is necessary and is supported by the full expressive power of the UML, possibly augmented by other profiles such as the EDOC profile [40].

**ESMS** An application offering a enterprise-oriented management view of an electronic service environment. For example, the HP Service Composer [16], or the DySCo research prototypes [49]. Other candidate technologies might be an application service offering a middle-tier of business logic, with a web-server providing the management interfaces.

**WfMS** A workflow management system, either embodying a capability (enactment) or coordinating a number of subcapabilities. Examples of workflow applications are IBM's MQ-Series Workflow [17] and PeopleSoft's [47] PeopleTools and Integration Broker.

**ERPS** An Enterprise Resource Planning System, dedicating to coordinating entities in the system, presumably making them available to fulfil capability roles. We do not consider resource planning in this paper, although

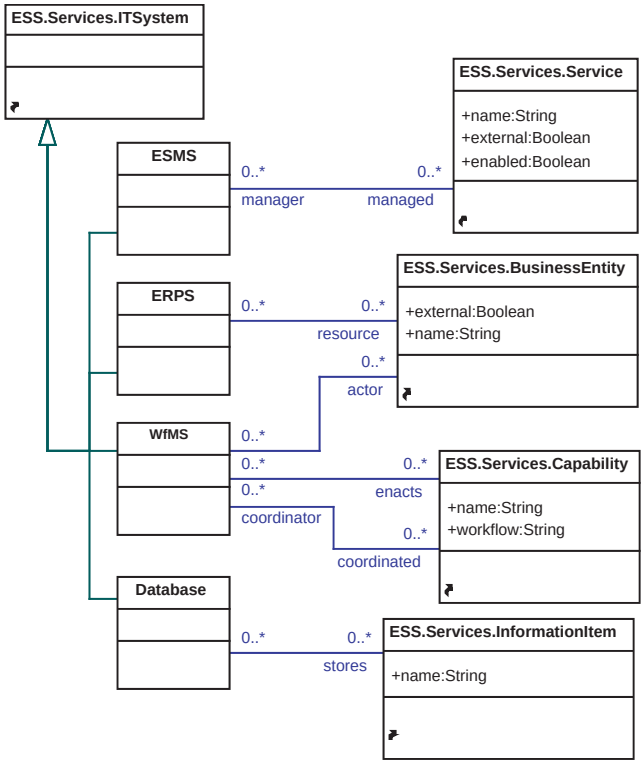


Figure 5.4: The management meta-model

it interacts at a functional level with coordination based on capabilities, and future work may provide a combined modelling approach. Examples of ERP systems are SAP's mySAP [53] and Baan's iBaan [3].

**Database** Most enterprises use databases to store information about the enterprise. Establishing a relationship between the (conceptual) information items and the databases that store them allows a modeller to check whether the information required by a business entity to fulfil a capability role is available in its context. Popular databases are Oracle [45] and MySQL [35].

## 5.3 The ESS Profile

The following tables relate elements in the meta-model to profile elements and elements in the UML meta-model.

All name attributes in the meta-models map to the name attribute of the class element in the UML meta-model. All associations in the meta-model

| Meta-model element | Stereotype | UML base class | Parent | Tags |
|---|---|---|---|---|
| Service | Service | Class | – | external<br>enabled |
| Service.content | content | AssociationEnd | – | – |
| Service.provisioning | provisioning | AssociationEnd | – | – |
| Service.component | component | AssociationEnd | – | – |
| Capability | Capability | Class | – | |
| Capability.input | input | AssociationEnd | – | – |
| Capability.output | output | AssociationEnd | – | – |
| CapabilityRole | CapabilityRole | Class | – | – |
| InformationItem | InformationItem | Class | – | – |
| Observation | Observation | Class | – | condition<br>observation |
| Observation.input | input | AssociationEnd | – | – |
| Observation.output | output | AssociationEnd | – | – |
| BusinessEntity | BusinessEntity | Class | – | – |
| ServiceOffer | ServiceOffer | Class | – | enabled |
| ServiceImplementation | Fulfills | Association | – | service |
| ITSystem | ITSystem | Class | BusinessEntity | – |
| ESMS | ESMS | Class | ITSystem | – |
| WfMS | WfMS | Class | ITSystem | – |
| WfMS.actor | wfactor | AssociationEnd | – | – |
| WfMS.enacts | enacts | Class | – | – |
| WfMS.coordinated | coordinates | Class | – | – |
| ERPS | ERPS | Class | ITSystem | – |
| Database | Database | Class | ITSystem | – |

Table 5.1: Stereotypes in the ESS profile

map to associations in models. Stereotypes on AssociationEnds are used to disambiguate associations where more than one exists between the same two meta-model elements. The meta-model constraints also have translations into constraints on the profile elements, and additional constraints reflect the structure of the original meta-model. For example, the 'Fulfills' stereotype can only be attached to an association between a CapabilityRole and a BusinessEntity, and its service tag must always be present:

**package** Foundation::Core
  **context** Association
    **inv**:

| Meta-model element | Tag | Stereotype | Type | Multiplicity |
|---|---|---|---|---|
| Service.external | external | Service | Boolean | 0..1 |
| Service.enabled | enabled | Service | Boolean | 0..1 |
| Capability.workflow | workflow | Capability | String | 0..1 |
| Observation.condition | condition | Observation | String | 1 |
| Observation.observation | observation | Observation | String | 1 |
| ServiceImplementation.service | service | Fulfills | Class | 1 |
| BusinessEntity.external | external | BusinessEntity | Boolean | 0..1 |
| ServiceOffer.enabled | enabled | ServiceOffefr | Boolean | 0..1 |

Table 5.2: Tags in the ESS profile

self.stereotype→exists("Fulfills") **implies**
self.connection.participant.stereotype→exists("CapabilityRole")
**and**
self.connection.participant.stereotype→exists("BusinessEntity")
**and**
self.taggedValue.type→exists(name = "service")

## 5.4 Example

We now present an example of the profile in use to model a freight moving service, based on previous research in the freight domain [31]. This example was also used to demonstrate the HP service composer.

Figure 5.5 shows the freight service and the capabilities that support it. The service is provisioned by a tendering capability. This service bids in a reverse auction. Simultaneously it coordinates resources required for the freight movement. Resource are traded in an online market in order to drive down the overhead of the transport. Details of this process are covered by the workflow description for these capabilities, not shown here.

A successful tender resorts in a move order. This is an example of information in the ESS. The workflow order is stored in the orders database. The profile combines conceptual and deployment elements, such as information items and databases to combine a logical view of the ESS with the underlying resources.

Each capability has an associated workflow description. The workflow for the handover capability is shown using an informal notation. The role names present correspond to associated capability roles.

The combination of conceptual view and resource view is also in evidence in

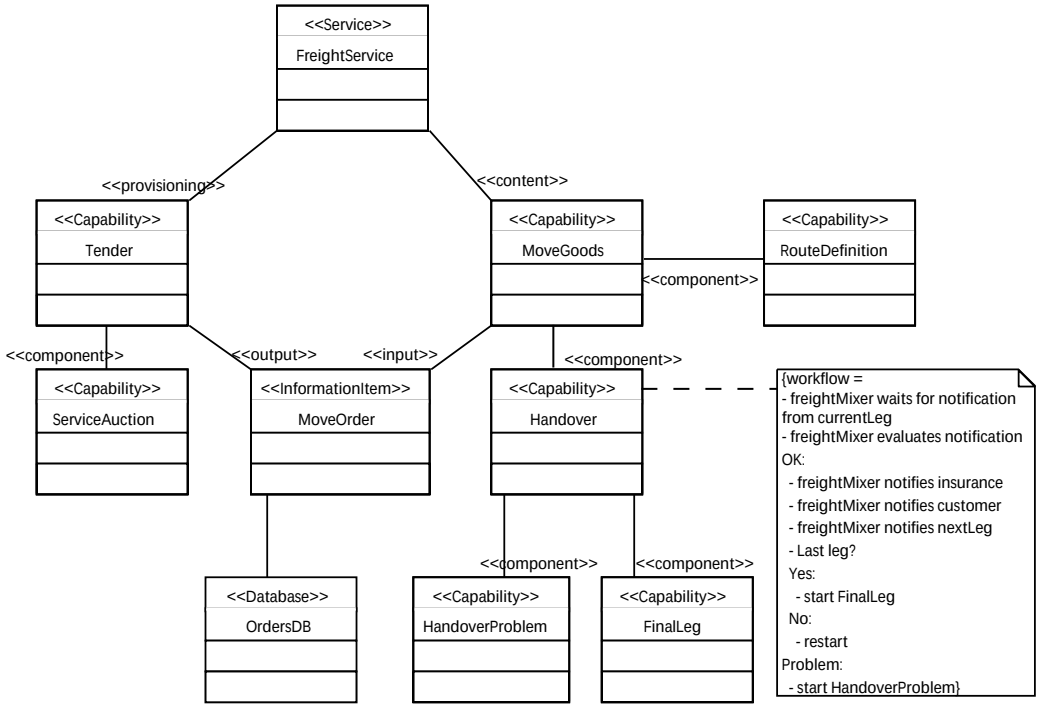


Figure 5.5: Services and capabilities in the Freight-mover example

Figure 5.6. This model shows the association of commercial entities with the roles associated with the handover capability.

## 5.5 Related work

The definition and characteristics of the ESS model derive substantially from the experience of HP Service Composer. UML notation is used in the HPSC, with a separation between platform-dependent and platform-independent models of an electronic service. Workflow notation and technology is used to model and manage the business logic of a service.

The ESS model is also closely related to the DySCo (Dynamic Service Composer) [49] research prototype. DySCo is the result of a two-year project involving University College London (UK), the University of St. Petersburg (Russia), the University of Ferrara (Italy), the University of Hamburg (Germany), and Hewlett-Packard (UK and USA). The objective of DySCo was the development of a conceptual and technology framework for the dynamic composition of electronic services. While lacking direct support for UML, DySCo provides

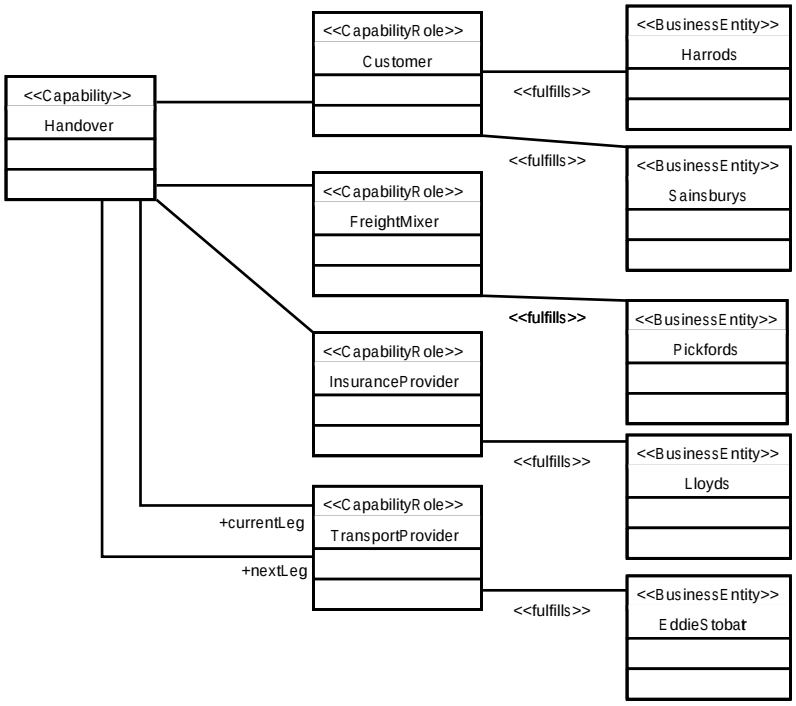


Figure 5.6: Resource planning in the Freight-mover example

modelling facilities for workflows and a homogeneous execution platform for an ESMS.

An electronic-services model is currently being used in the context of the EGSO (European Grid for Solar Observations) [4] project. The model-driven approach to the architecture of the service provision part of the EGSO grid is expected to address the need to integrate services based on different provision models and execution platforms. Each service provider in the EGSO grid will be equipped with an ESMS. In addition, a specific ESMS federates and manages the service provisioning capabilities of the overall EGSO grid.

The Enterprise Collaboration Architecture (ECA) defined in the OMGs EDOC specification [40] provides a comprehensive framework for the modelling of enterprise systems. The ESS profile introduces enterprise system components that can be designed based on the ECA, and provides a means to model features peculiar to electronic services that are not explicitly addressed by the ECA. Similar considerations apply for the Reference Model for Open Distributed Processing (RM-ODP) [19], which is also closely related with the ECA.

Most technology and conceptual frameworks for electronic services [25] focus

on web-service-based automation of the front-end of individual services. Web Services [8, 10] constitute the reference model for access to and basic orchestration of business resources. We envision Web Services playing a fundamental role in the realisation of electronic services. Still, a more comprehensive approach is needed for the realisation and operation of business-level services. An example of the issues involved in the realisation of business-level service is HiServs Business Port [23]. FRESCO (Foundational Research on Service Composition) [50] provides an example of second-generation framework for electronic service management. The focus of FRESCO is on the provision aspects of services.

# Chapter 6

# Conclusions and Future Work

## 6.1 Conclusions

In this document we have presented an overview of our approach to service composition and analysis. The method relies on the integration of analysis methods, representations of QoS aware systems, and SLA specifications in UML design tools. This places all the information required to reason about service composition and analysis in a single repository. The reasoning itself is supported by model transformations between design and analysis models, performed within the same repository. This enables a powerful and flexible approach to analysis.

We have also demonstrated the use of UML and OCL to precisely define the meaning of service-level agreements in our language SLAng. This significantly enhances the utility and credibility of the language.

Finally, through the example of electronic service systems we have provided an example of the modelling of a service architecture in order to support the development of such systems according to an MDA approach. We feel that this approach is complementary to the TAPAS project, and believe that applying such a modelling effort to the TAPAS architecture once finalised will yield both concise documentation of the architecture and valuable modelling support for its adopters.

## 6.2 Future Work

### 6.2.1 Analysis of designs

In Chapter 3 we presented a general framework for associating analysis models with designs. The analysis models exhibit the vocabulary and structure of particular mathematical formalisms, for example, queuing networks, Petri-nets or Bayesian networks. Each formalism is appropriate for determining a few QoS properties, such as performance or reliability. The design models potentially incorporate architectural features, such as the profile elements introduced in Chapter 5 for electronic-service systems, the EJB profile [20], or the EDOC profile [40]. Clearly the benefit of the approach depends on establishing mappings between as many architectures and analysis models as possible.

Estabilishing a mapping that is guaranteed to produce valid and feasible analysis models for a particular architecture is a significant challenge, as dicussed in Section 3.3. We will therefore focus on providing the most useful analyses for the most relevant architectures. Candidates for future investigation will be an investigation into the performance and reliability of EJB servers, and the modelling of similar properties for the TAPAS architecture. As previously stated, the TAPAS architecture should be designed in such a way to provide easily predictable QoS characteristics.

### 6.2.2 SLAng

SLAng is an evolving language. The related work section highlights desirable features of previous SLA languages that SLAng could benefit from incorporating. These include: Payments related to service violations; reuse features, including SLA templates, clause reuse and generalisation hierarchies; the specification of management actions, performed in response to QoS state changes or SLA violations; the ability to define new parameter types in terms of existing types; the ability to distribute clauses to third parties without exposing sensitive details of the client and server; and the ability to define dynamic relationships between service levels, effectively defining the permissible states for a system capable of adapting its QoS behaviour.

These features would enhance the utility and usability of the language without modifying the semantic definition of the underlying parameters, although potentially requiring extensions to the existing semantics. Additional work will further realise the value of the semantic definition: By automating consistency checking of SLAs; by implementing application monitors that rely on the definition of SLAs to assess conformance to SLAng terms; by defining additional

relationships between SLAs, and automating their comparison; and by investigating methodologies and designs for systems governed by SLAng SLAs.

### 6.2.3 Modelling the TAPAS architecture

It is our intent to apply the approach described in Chapter 5 to develop both a normative model of the TAPAS architecture and a modelling language based on that domain. Components of the domain will be architectural components from the TAPAS platform, protocol elements required to negotiate service usage and business elements supporting service composition. The intent of the modelling language will be to support the planning and development of trusted and QoS aware systems according to an MDA development strategy.

### 6.2.4 General

Our method relies on extending UML with a variety of domain specific languages each appropriate to a modelling task, for example, performance analysis or the representation of SLAs. Each language is defined seperately, but the common basis in UML permits the integration of all of these models into a coherent view of the system.

Each language has semantic information associated with it. SLAng, the profile for electronic service systems, and the real-time profile used as the starting point for analysis, all have rich semantic models. Other languages such as the profile for queuing networks have a definition based in mathematical formalisms.

We also define relationships between our language extensions. The mappings introduced in Chapter 3 relate analysis models to design domains. The correspondance stereotype introduced in Chapter 4 allows the association of QoS information of different types.

As the method is developed, there will be an increasing need to manage the interactions between domain specific languages. Our future research in this area is aimed at producing a approach to integrating multiple design languages in order to ensure the consistency of designs with respect to the semantic definitions of the languages and their inter-relationships.

# Bibliography


[1] Specifying and measuring quality of service in distributed object systems. In *1st International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 43 – 52, Kyoto, Japan, April 1998. IEEE Press.

[2] *WOSP 2002, Third International Workshop on Software and Performance.* ACM Press, 2002.

[3] Baan. *iBaan.* `http://www.baan.com/`.

[4] R. D. Bentley. EGSO – the european grid of solar observations. In *European Solar Physics Meeting, ESA Publication SP-506*, 2002.

[5] A. Bertolino and R. Mirandola. Modelling and analysis of non-functional properties in component-based systems. In *Workshop on Test and Analysis of Component Based Systems (TACoS '03), in conjunction with ETAPS '03*, Electronic Notes in Theoretical Computer Science (ENTCS), Warsaw, Poland, April 2003. Elsevier Science B. V.

[6] J. Bodeveix, T. Millan, C. Percebois, C. L. Camus, P. Bazex, L. Feraud, and R. Sobek. Extending OCL for verifying UML models consistency. In *[26]*, 2002.

[7] G. Brahnmath, R. R. Raje, A. Olson, M. Auguston, B. R. Bryant, and C. C. Burt. A quality of service catalogue for software components. In *Southeastern Software Engineering Conference*, pages 513 – 520, Huntsville, Alabama, USA, April 2002.

[8] E. Cerami. *Web Services Essentials.* ORielly and Associates, 2002.

[9] Computer Science Research Laboratory, "BABES-BOLYAI" University, Romania. *OCL-Evaluator.* `http://lci.cs.ubbcluj.ro/ ocle/`.

[10] M. C. et Al. *Web Services Business Strategies and Architectures.* Expert Press, 2002.

[11] A. S. Evans and S. Kent. Meta-modelling semantics of uml: the puml approach. In *2nd International Conference on the Unified Modeling Language*, volume 1723 of *Lecture Notes in Computer Science (LNCS)*, pages 140 – 155. Springer-Verlag, 1999.

[12] G. Ferrari and G. Lodi. TAPAS architecture – QoS enabled application servers. Technical report, The TAPAS Project, March 2003.

[13] D. Frankel. *Model Driven Architecture - Applying MDA to Enterprise Computing*. OMG Press. Wiley Publishing, Inc., 2003.

[14] S. Frolund and J. Koistinen. Qml: A language for quality of service specification. Technical Report TR-98-10, Palo Alto, California, USA, 1998.

[15] B. Gibb and S. Damodaran. *ebXML: Concepts and Application*. John Wiley and Sons, 2002.

[16] Hewlett-Packard Company. *HP Service Composer User Guide*, 2002.

[17] IBM. *Websphere MQ Workflow*. `http://www-3.ibm.com/ software/ integration/ wmqwf/`.

[18] International Business Machines (IBM), Inc. *Web Service Level Agreement (WSLA) Language Specification*, January 2003.

[19] ISO/IEC, ITU-T. *Open Distributed Processing – Reference Model – Part 2: Foundations, ISO/IEC 10746-2, ITU-T Recommendation X.902*.

[20] Java Community Process. *UML/EJB Mapping*, August 2001. `http://www.jcp.org/ jsr/ detail/ 26.jsp`.

[21] N. Kaveh and W. Emmerich. Deadlock detection in distributed object systems. In *Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9)*, Vienna, Austria, 2001. ACM Press.

[22] A. Kleppe and J. Warmer. *Unification of Static and Dynamic Semantics of UML*. Klasse Objecten. `http://www.klasse.nl/ english/ uml/ uml-semantics.html`.

[23] R. Klueber and N. Kaltenmorgen. eServices to integrate ebusiness with ERP systems – the case of HiServs business port. In *Workshop on Infrastructures for Dynamic Business-to-Business Service Outsourcing (CAISE-ISDO)*, 2000.

[24] P. Krutchten. *The Rational Unified Process: An introduction*. The Object Technology Series. Addison Wesley, 1999.

[25] H. Kuno. Surveying the e-services technical landscape. In *Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS)*. IEEE Press, 2000.

[26] L. Kuzniarz, G. Reggio, J. Sourrouille, and Z. Huzar, editors. *Workshop on Consistency Problems in UML-based Software Development, UML 2002*. Department of Software Engineering and Computer Science, Blekinge Institute of Technology, 2002.

[27] D. D. Lamanna, J. Skene, and W. Emmerich. D2: Specification language for service level agreements. Technical report, The TAPAS Project, March 2003.

[28] D. D. Lamanna, J. Skene, and W. Emmerich. Specification language for service level agreements. Technical Report D2, University College London, March 2003.

[29] K. Lano, D. Clark, and K. Androutsopoulos. Formalising inter-model consistency of the UML. In *[26]*, 2002.

[30] E. Lazowska, J. Zahorjan, G. S. Graham, and K. Sevcik. *Quantitative System Performance*. Prentice Hall, Inc., 1984.

[31] N. Linketscher and M. Child. Trust issues and user reactions to e-services and e-marketplaces: A customer survey. In *DEXA Workshop on e-Negotiation*, 2001.

[32] J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. John Wiley and Sons, 1999.

[33] A. Marton, G. Piccinelli, and C. Turfin. Service provision and composition in virtual business communities. In *IEEE-IRDS Workshop on Electronic Commerce*, Lausanne, Switzerland, 1999.

[34] W. E. McUmber and B. H. C. Cheng. A general framework for formalizing UML with formal languages. In *Proceedings of the 23rd international conference on Software engineering*. IEEE Computer Society, 2001.

[35] MySQL AB. *MySQL Database*. `http://www.mysql.com/`.

[36] The Object Management Group (OMG). *Joint Submisison to the QoS for Fault Tolerance RFP, I-Logix, Open-IT, and THALES*, realtime/03-08-06 edition.

[37] The Object Management Group (OMG). *The CORBA Trading Service*, formal-97-04-01 edition, May 1997.

[38] The Object Management Group (OMG). *Model Driven Architecture (MDA)*, ormsc/01-07-01 edition, July 2001.

[39] The Object Management Group (OMG). *The Meta-Object Facility v1.4*, formal/2002-04-03 edition, April 2002.

[40] The Object Management Group (OMG). *UML Profile for Enterprise Distributed Object Computing Specification*, ptc/02-02-05 edition, May 2002.

[41] The Object Management Group (OMG). *XML Metadata Interchange (XMI), version 1.2*, formal/02-01-01 edition, 2002.

[42] The Object Management Group (OMG). *UML Profile for Schedulability, Performance and Real-time Specification, Final Draft*, ptc/03-03-02 edition, March 2003.

[43] The Object Management Group (OMG). *UML Profile for Schedulability, Performance, and Time Specification*, ptc/03-03-02 edition, 2003.

[44] The Object Management Group (OMG). *The Unified Modelling Language v1.5*, formal/2003-03-01 edition, March 2003.

[45] Oracle. *Oracle database products*. `http://www.oracle.com`.

[46] B. Pagurek, V. Tosic, and K. Patel. Reusability constructs in the web service offerings language (wsol). In *Workshop on Web Services, e-Business, and the Semantic Web '03 (CAiSE-WES)*, Lecture Notes in Computer Science (LNCS), Velden, Austria, June 2003. Springer-Verlag.

[47] PeopleSoft. *PeopleTools and Integration Broker*. `http://www.peoplesoft.com/`.

[48] D. Petriu and H. Shen. Applying the UML performance profile: Graph grammar-based derivation of LQN models from UML specifications. In *Proceedings of Performance TOOLS 2002*, 2002.

[49] G. Piccinelli and L. Mokrushin. Dynamic e-service composition in DySCo. In *Workshop on Distributed Dynamic Multiservice Architecture, IEEE ICDCS-21*, Phoenix, Arizona, USA, 2001.

[50] G. Piccinelli, C. Zirpins, and W. Lamersdorf. The FRESCO framework: An overview. In *Symposium on Applications and the Internet (SAINT), IEEE-IPSJ*, 2003.

[51] R. Pooley. Using UML to derive stochastic petri net models. In *Proceedings of the fifteenth UK Performance Engineering Workshop (UKPEW)*, pages 45–56, 1999.

[52] G. N. Rodrigues, G. Roberts, W. Emmerich, and J. Skene. Reliability support for the model driven architecture. In *Workshop on Software Architectures for Dependable Systems (ICSE-WADS)*, Portland, Oregon, USA, May 2003. ACM Press.

[53] SAP. *mySAP*. `http://www.sap.com/`.

[54] J. Skene and W. Emmerich. Full example profiles. `http://www.cs.ucl.ac.uk/ staff/ j.skene/ FSE-2003-profiles`.

[55] J. Skene and D. D. Lamanna. Semantics of SLAng ASP SLAs. `http://www.cs.ucl.ac.uk/ staff/ j.skene/ tapas/ asp.doc`.

[56] C. Smith and L. Williams. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Object Technology Series. Addison Wesley, 2001.

[57] R. Staehli, F. Eliassen, J. O. Aagedal, and G. Blair. Quality of service semantics for component-based systems. In *2nd International Conference on Reflective and Adaptive Middleware Systems - Middleware 2003*, volume 2672 of *Lecture Notes in Computer Science (LNCS)*, pages 153 – 157. Springer-Verlag, June 2003.

[58] F. Steimann and T. Khne. A radical reduction of UML's core semantics. In *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*. Springer, 2002.

[59] Sun Microsystems, Inc. *Java Metadata Interface JMI specification*. `http://java.sun.com/ products/ jmi/`.

[60] Sun Microsystems, Inc. *Enterprise Java-Beans (EJB) Specification v2.0*, August 2001.

[61] V. Tosic, B. Esfandiari, B. Pagurek, and K. Patel. On requirements for ontologies in management of web services. In *Workshop on Web Services, e-Business, and the Semantic Web '02 (CAiSE-WES)*, volume 2512 of *Lecture*

*Notes in Computer Science (LNCS)*, pages 237 – 247, Toronto, Canada, May 2002. Springer-Verlag.

[62] V. Tosic, B. Pagurek, K. Patel, B. Esfandiari, and W. Ma. Management applications of the web service offerings language (wsol). In *15th International Conference on Advanced Information Systems Engineering (CAiSE)*, volume 2681 of *Lecture Notes in Computer Science (LNCS)*, pages 468 – 484, Velden, Austria, June 2003. Springer-Verlag.

[63] UDDI.org. *UDDI (Universal Description, Descovery and Integration) Executive White Paper*, November 2003. `http://www.uddi.org/ pubs/ UDDI_Executive_White_Paper.pdf`.

[64] The World Wide Web Consortium (W3C). *Web Services Description Language (WSDL) 1.1.*