This is my Master Thesis, submitted for the Master of Science in Informatics at the Department of Informatics, University of Oslo. The thesis has been a project at the department of Telecom and Informatics at SINTEF, Oslo.
I would like to thank my supervisors Arne-Jørgen Berre and Hjørdis Hoff for always giving me support, encouragement and deadlines. I would also like to thank my fellow students Kai Fredriksen and Berge Stillingen for always being ready to participate in discussion.
And to Christel, thank you for letting me work when I needed, and always giving me love, encouragement and patience

Abstract

Creating software systems by composing already existing reusable software is a vision which has been driving the development of software technologies and paradigms for a long time. By combining visual modelling and service oriented architecture, this thesis proposes a visual language for composition of heterogeneous service, called AuSCL (Another unified Service Composition Language).
The thesis presents requirements for service composition in general, and additional requirements introduced by the use of heterogeneous service technologies. Existing visual languages such as UML2 and BPMN has been investigated and evaluated in three case studies. This has lead to a list of potential improvements for UML2 and BPMN which have been used in the design of AuSCL.
AuSCL is a UML2 profile, introducing a set of stereotypes to enhance UML2 functionality and a domain specific structure of model views for modelling a heterogeneous service composition from a set of viewpoints. This structure consists of a set of model views and are introduced to narrow the extensive modelling possibilities provided by UML2. The model views are divided into abstract and concrete views, and does also separate between internal and external aspects. AuSCL extends UML2 to support dynamic service selection (service discovery and and runtime selection) for late binding and a consistent way of combinig activities and interactions to model communication.
AuSCL is evaluated against the identified requirements and by implementation of the same case studies used in the evaluation of UML2 and BPMN. The evaluation shows that AuSCL is better suited than UML2 and BPMN for visual modelling of heterogeneous service composition for the identified requirements and case studies.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Programming by reusing already created software has been a vision in the software community for a long time: in 1968 M.D. McIlroy said at the NATO conference [1]:

> We say ... Not, "What mechanism shall we use?" but "What mechanism shall we build?" I claim we have done enough of this to start taking such things off the shelf.

McIlroy discussed routines used when creating compilers in this quote, but the principle of building software by composition of existing parts of software taken "off the shelf" is still highly relevant. In 1992 Megaprogramming [2] was introduced as a process and a programming language (CHAIMS) for programming by defining high-level composition of coarse grained megamodules that capture the functionality of services provided by large organizations.
Programming by composition has been one of the driving forces for many programming paradigms and methods, such as object orientation, component orientation and service orientation.
Creating reusable pieces of software that can be wired together to create new applications is the target. This is difficult to achieve due to the complex nature of software systems. Service oriented architectures tries to solve this by defining the services to be process oriented, loosely coupled and specified by a functional description, thus reducing the complexity of the dependencies between the services.

SODIUM SODIUM (Service-Oriented Development In a Unified fraMework) [3] is a current research project funded by the European Commission which aims to support the standards-based unified composition of heterogeneous p2p, grid and web services. The work described in this thesis was performed as a part of the Sodium project.
Figure 1.1 shows the architecture of the SODIUM composition suite application. The SODIUM platform is a complete platform for visual composition, service discovery, transforming and executing a service composition of heterogeneous services. The composition suite consists of three main parts, the visual modelling

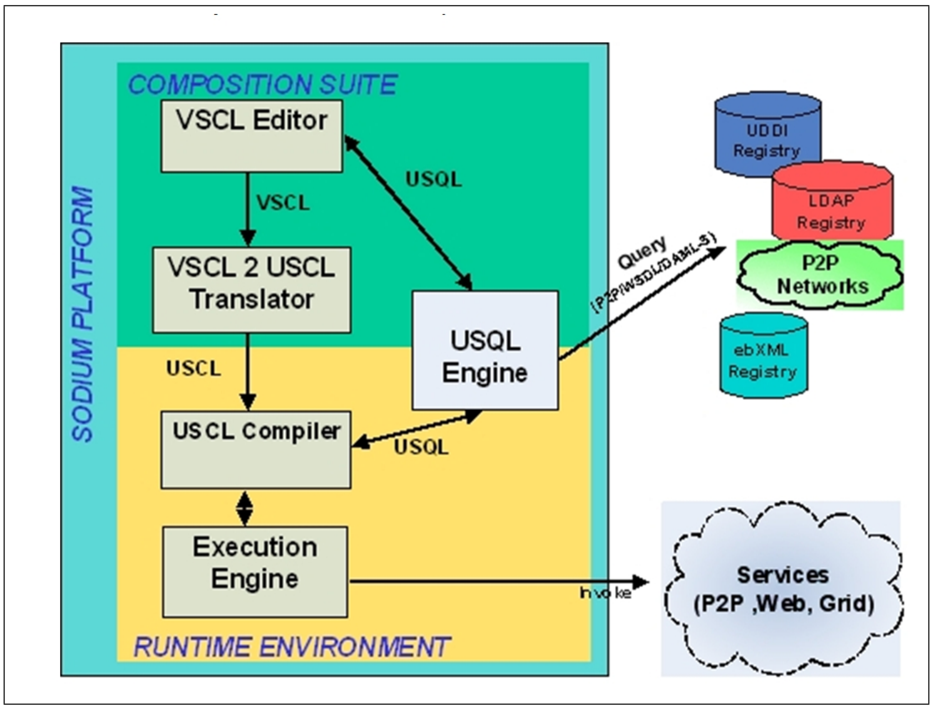


Figure 1.1: Sodium architecture.

language (VSCL), the execution environment (USCL) and a query mechanism for discovering services (USQL).

1.1 Motivation

Programming by composition can potentially reduce the development time and cost of software systems. Programming by composition requires coarse grained, loosely coupled parts (objects, components or services) to reduce the complexity of connecting the parts to other parts when creating the new application. Fine grained parts creates dependencies between the parts of a composition making it harder to change or swap the parts at a later stage.
From a developers perspective, visual languages are more suitable for specifying compositional systems than lexical languages, while lexical languages are more suitable for executing the composition. Two new technological innovations create a suitable environment for visual composition of software systems:

- Service Oriented Architectures (SOA)[4], focusing on loosely coupled functional units of software. The service are accessed and described only through its interface as all other implementation details of the service is irrelevant to the the service consumer.
- Model Driven Architectures (MDA) [5] specifying the system visually in a formal modelling language, focusing on creating platform independent models of the system.

Service Oriented Architecture Services are coarse grained functional units of software, implementing a contractually defined interface. The services are stateless, they have no pre-requisites in terms of behaviour for the client. The client does not have to perform any operation before consuming the service. New value-added composite services can be created by composing a set of already existing services.
Technologies such as the Web Service stack of standards, SOAP[6], WSDL[7], UDDI[8], BPEL[9], makes it possible to create platform-independent services, using the Internet as the communication channel, and XML to enhance interoperability between programming languages.

Heterogeneous Services Several technologies are being created for implementing services, running on different platforms such as P2P network and grids. These new technologies are different from the Web Service technologies in some aspects, either in the technologies they used for service consumption, or in the paradigm they have for designing services. These differences in technology and paradigms create challenges in the composition process, which are not relevant if all service are based on Web Services.

Model Driven Architecture Model Driven Architecture is a framework for using models to develop software systems. By using a formal modelling language and creating models at different abstraction levels with transformation technology to automatically create a model based on the higher level model, executable software systems can be created on the bases of models. The MDA approach creates platform independent models, describing the system without binding it to a specific platform, creating portable and interoperable system definitions.
Programming by composition is an excellent candidate for a model driven approach, visually representing the parts of the composition in a unified platform independent model, making it possible to create compositions of parts without considering the platform specific details, as there is a unified visual model of the part being used.

Existing Visual Modelling Languages for Service Composition Several visual modelling languages exists today. Some, like UML2[10] is very open, and can be used to model most kind of software systems but does not have any specific features available for heterogeneous service composition. Others, like BPMN[11], is domain specific, and is tailored for a specific purpose, executable business processes in the case of BPMN, but lacks the support for heterogeneous services. A visual language that is specialized for the requirements related to composition of heterogeneous services, and has the necessary tool support is not available today. Another interesting candidate, the BPDM notation [12] is not supported by any available tool, while UML4EDOC [13] is only implemented in UML1.X and not in UML2.

Business Process Management Orchestrating services to create new composite services that are high-level and coarse grained, makes it possible to have services which represents business-objectives. These services can be combined with a business process definition to create automated business processes. This high level representation of the software system, can be easily mapped to business level models, and reduce the differentiation between the business and the software system. The system will be naturally hierarchical as the business processes will consist of composite services, which again consists of simpler services. At the top level the business processes in the separated companies will be communicating by exchanging messages. This will create a process-driven service oriented architecture, with emphasis on composition of reusable services.

1.2 Scope of thesis

This thesis will consider visual composition of heterogeneous service. Semantic description of services, and the use of semantic information in composing will not be considered in this thesis. Quality of Service aspects of service selection and management is not in the scope of this thesis.
In this thesis the heterogeneous service technologies used are Web Services, grid based services and P2P-network based services.

1.3 Research Challenges

The research challenges investigated and evaluated in this thesis are:

- There are actual requirements for visual composition of heterogeneous services, that are not supported by existing solutions.
- No visual service composition language available today is suitable for modelling composition of heterogeneous services.
- A new language called AuSCL will be suitable for modelling heterogeneous service composition.

1.4 Method

This section presents the methods used to investigate and answer the research questions. Several approaches have been used, both experimental work and a study of the current state of the art.

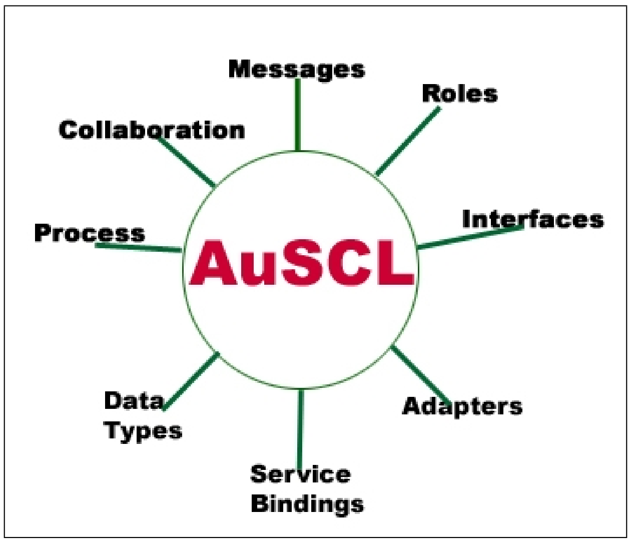
Identification of visual heterogeneous service composition requirements By studying existing languages and implementing cases using these languages, requirements for visual composition of heterogeneous services were identified. Other technologies related to service composition and heterogeneous services has also been investigated. This has identified requirements that are specific to the service implementation technologies.

Case based evaluation of existing solutions The study of existing languages identified two languages which was studied further in a case based evaluation, UML2 and BPMN. Using a case based evaluation gave a more practical viewpoint, and made it easier to find advantages and shortcomings in relation to real world problems. By using a case with a heterogeneous set of service implementation technologies it was possible to understand if this introduces some requirements that are not present in a homogeneous environment using only Web Services. A description of the cases used can be found in chapter 4, while the case implementations can be found in appendices C, D and E.

Development of a specification for Another Unified Service Composition Language Using the state of the art study, the specified requirements and the practical experience of implementing cases with the most promising existing solutions as a base for a proposed solutions to address the identified shortcomings.

Case based evaluation of AUSCL To evaluate the proposed solution and compare it with the existing solutions it is important to use the same case studies as in the evaluation of the existing solutions. This also highlighted the enhanced capabilities of the new proposed solution. A description of the cases used can be found in chapter 4, while the case implementations can be found in appendices C, D and E.

1.5 AuSCL



In chapter 6 the UML2 profile "Another unified Service Composition Language" is presented. This language extends UML2 to enhance it's capabilities in modelling heterogeneous service compositions. AuSCL uses a set of model views, and a method to structure these views. The structure focuses on separation of abstract and concrete information, and a separation of concerns, where each model-view is used to model a specific aspect of the model.
The abstract parts of the model consists of these model-views:

- Interfaces Model View - Definition of the interfaces used in the service composition.
- Messages Model View - The messages passed between the actors in the service composition.
- Roles Model View- A collection of interfaces, both provided and required, defines a role.
- Collaboration Model View - Definition of behavior between partners.
- Process Model View - The orchestration of services that define the behavior of the service composition.

The concrete parts of the model binds the abstract parts of the service composition the concrete services. This is either done directly or dynamically, performing a query to a service broker. The model-views in the concrete parts of the model are.

- ServiceBindings Model View - Binding of abstract roles to concrete services.
- Datatypes Model View - Definition of data in the messages.
- Adapters Model View - A possible model-view to increase the decoupling between partners, by adding an adapter between the process and a concrete service.

Main contributions The main contributions of the AuSCL proposal are:

- A domain specific structure for a set of model views to represent a heterogeneous service composition model. These structure has a clear separation of concern, with a separation between concrete and abstract model views, and between internal and external aspects of the model. The structure also increases modularity of the model. The structure narrows doen the options given in UML2 and helps the modeller in creating a service composition model.
- A set of stereotypes to enhance UML2's capabilities in modelling heterogeneous service composition. These stereotypes introduces dynamic service selection for late binding and increases the ability to model consistently the communication with an external partner, by combing sequence and activity diagrams, at different abstraction levels.

1.6 Structure of thesis

This thesis is contains the following chapters.

- Introduction - This chapter contains the motivation for the work performed, the research challenges and method used for evaluating these hypotheses.
- Background - This chapter includes a explanation of the central concepts in the domain, as well as a overview of selected visual languages and other related work.
- Requirements - Here the requirements for a visual language for composition of heterogeneous services are presented.
- Case Descriptions - All cases used in this thesis are described here, as well as mapped to the relevant requirements.
- Evaluation of Existing Solutions - This chapter presents an evaluation of UML2 and BPMN based on the discovered requirements and the case implementations that have been done.
- Another unified Service Composition Language - A presentation of the proposed UML2 profile for composition of heterogeneous services.
- Evaluation of AuSCL - This chapter contains an evaluation of AuSCL and a comparison of AuSCL with UML2 and BPMN. There is also a discussion of the research challenges.
- Conclusions and Future Work - Conclusions of the work presented in this thesis.
- Appendix A. Background - This appendix presents more details of some of the languages and technologies presented in chapter 2.
- Appendix B. Expressiveness Workpatterns - A presentation of workpatterns, some related work. The 5 basic workpatterns are detailed.
- Appendix C. Case A. Request for Proposal - This appendix contains a description of the case, and all implementation models.
- Appendix D. Case B. Generic Disk Drive- This appendix contains a description of the case, and all implementation models.
- Appendix E. Case C. Distributed Office Backup - This appendix contains a description of the case, and all implementation models.

Chapter 2

Background

This chapter presents some of the main concepts that are important for the work presented in this thesis. Topics in service oriented architectures as well as model driven architectures will be presented. Further on is a short presentation of visual modelling languages and their capabilities for service composition. This is followed by a presentation of other current work that is related to the work presented in this thesis. More details of the visual modelling languages can be found in appendix A.

2.1 Explanation of Central Concepts

2.1.1 Service Oriented Architecture

In a service oriented architecture the building block for creating applications are services. Functionality in systems and applications are made available as loosely coupled services, which are autonomous and coarse grained. Services has a clear separation of implementation and description, which makes it possible to implement services in a range of technologies, as the important aspect is the technology used for communication with the service. The term heterogeneous service relates to this and services with different service invocation paradigms or technologies.

2.1.1.1 Actors

[4] define three actors in a serviceinteraction; the service consumer, the service provider and the service broker, as can be seen in figure 2.1. A collaboration starts with the service provider publishing its service description, or service contract to the service broker. Then the service consumer sends a query to the service broker, and gets a response containing one or more service descriptions. These descriptions are used by the service consumer to bind to the concrete service instance and consume the service.

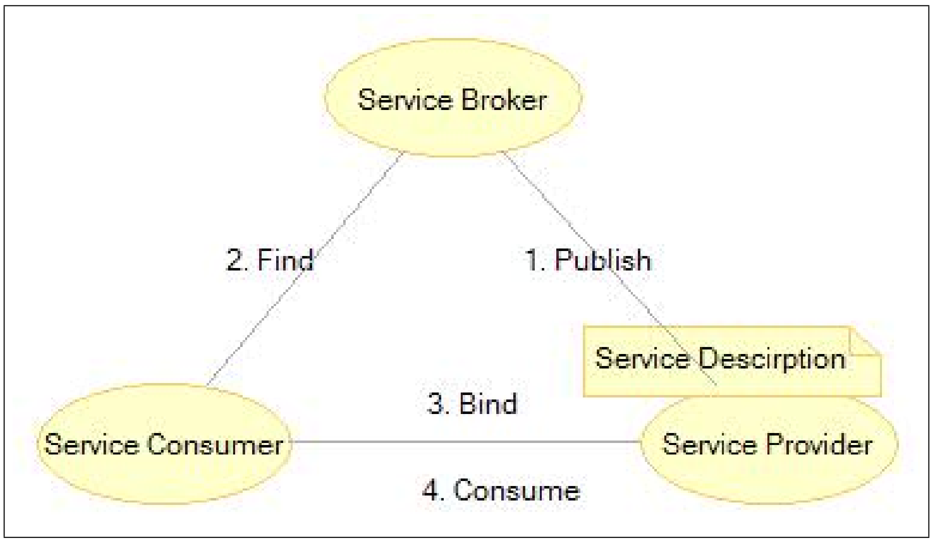


Figure 2.1: Service interaction

2.1.1.2 Service

Definition In the work described in this thesis a **service** is a functional unit that implements a contractually defined interface. A service should be coarse grained, document based and autonomous, thus representing the functionality of an application rather than a fine grained piece of business logic.

Description [4] states that service is not concretely defined, but states that from a coarse-grained point of view a service is an activity that is realized by an application, machine or human. In the context of application development, a service is a reusable building block that offers a particular functionality. [4] also state that a service offers functionality that is contractually defined in a service description, which contains a combination of syntactic, semantic and behavioral information. [14] has a similar view, and defines a service to be the externally observable behavior of an application. [15] talks about a service as a functional unit, and that it often corresponds with business functionality. It also mentions some key requirements for services in a service oriented architecture; high autonomy, coarse granularity and process awareness. A service can be stateless, where the service consumer only needs to invoke the service, and receive a response, conversational, where several message exchanges take place, and there is some protocol defining the order of the message exchanges, or stateful where the service consumer control the life cycle of a service, creating and destroying instances. The interaction pattern of a service can be seen in figure 2.1.

2.1.1.3 Heterogeneous Services

Several technologies and paradigms exists for a Service Oriented Architecture. Web Services are a set of standards for describing and invoking services using

XML. P2P services focus on the network architecture of the service implementation which is more dynamic in nature. Grid services uses stateful resources on a grid, which have operations for life cycle management and accessing properties.

Web Services Web Services are services that are described using WSDL [7], and uses the Internet as the network infrastructure. Normally SOAP [6] is used for implementation of the messages.

Technologies Web Services are a set of standards, that can be use to implement the discovery (UDDI [8]), description (WSDL) and invocation (SOAP) of services. Several other standards exists that extend the functionality of web services, adding support for security[16] and transactions[17], but the main properties for a service to be a web service is the usage of the WSDL and SOAP standards, as well as internet communication using HTTP. A Web Service is normally stateless, in that there is no need to do anything prior to invoking a Web Service, but some services are conversational, where a sequence of service invocations are necessary.

Peer-to-Peer Networks

Definition A peer-to-peer network is a network where there is no central place of control. Additionally the network use direct communication between peers, and peers both request and provide resources to other peers, and have the ability to handle peers going offline without losing network functionality.
A **p2p network service** is a service that gives access to the functionality of a peer-to-peer network application, this functionality can be implemented by a single peer, or a combination of peers. This service can access the peer-to-peer network functionality by becoming a peer in the network, acting as a gateway to the p2p network.
A **peer** is a member of a p2p network, and can communicate with all other peers in the network.

Description [18] defines a network architecture to be peer-to-peer if the participants share a part of their own hardware resources, and that these resources are necessary to provide the service of the network. Also any participant can be both a resource provider and a resource requester. The definition distinguishes between two types of a peer-to-peer network architecture; pure, where any terminal entity of the network can be removed without any loss of service-functionality, and hybrid, where a central entity is needed to provide the service. [19] focuses more on communication in its definition, stating that peer-to-peer computing is the sharing of computer resources and services through direct communication of systems.
Figure 2.2 shows an architectural model of a p2p-network. The network consist of several peers that each provide some services in the network. These services

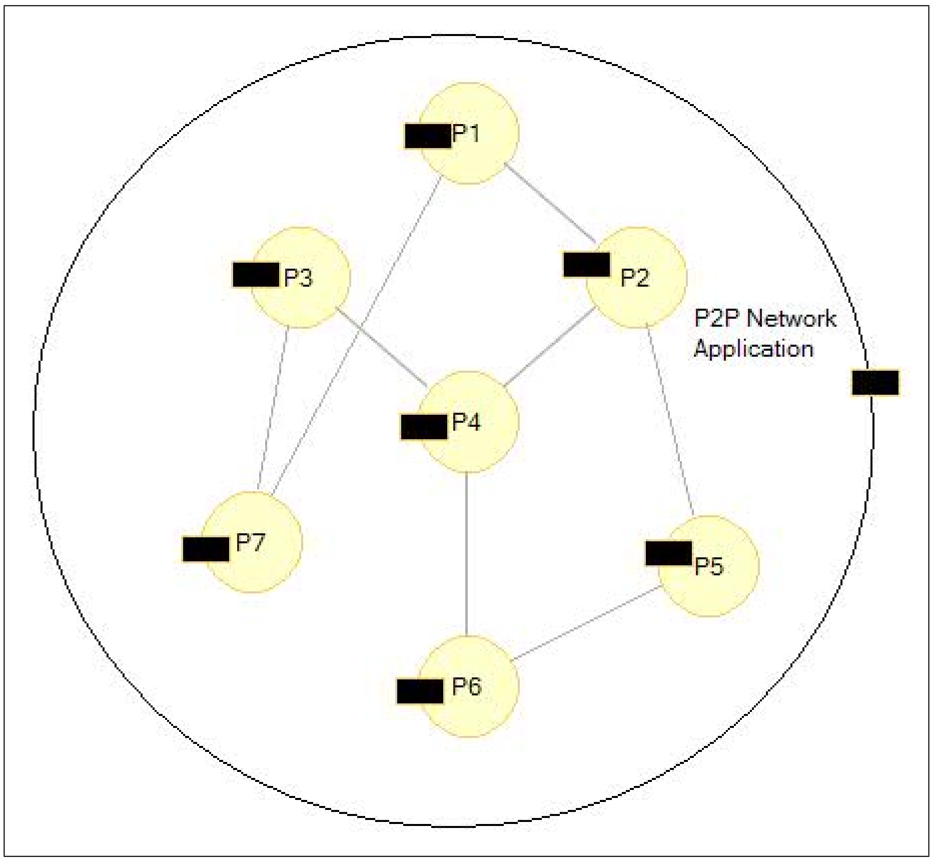


Figure 2.2: P2P network. The yellow circles represents peers, while the black rectangles represents a service interface.

can be invoked by any other peer on the network, but the often a search must be performed first to find the peer that provides the service. It is also possible for the network itself to provide some service, in the diagram depicted with the outer circle, as an p2p-network application, such as a file-search, which makes it possible to search the complete network for a file. A p2p network is suited for dynamic and unstable network environments, where the peers in the network are not always connected to the network. A p2p network keeps multiple copies of the resources distributed in the network. The most used technology for implementing p2p networks is the java based JXTA[20] technology. Applications such as Edutella[21] and Gnutella[22] is implemented using a P2P architecture.

Grid Services

Definition A **grid** means a set of interconnected distributed, possibly stateful, resources running on heterogeneous platforms. A **grid application** is an application that runs on a grid platform, implemented using WSRF [23]. A **grid service** is a service that is implemented on a grid, as a single resource, or an application using a set of resources. A grid application can be implemented by

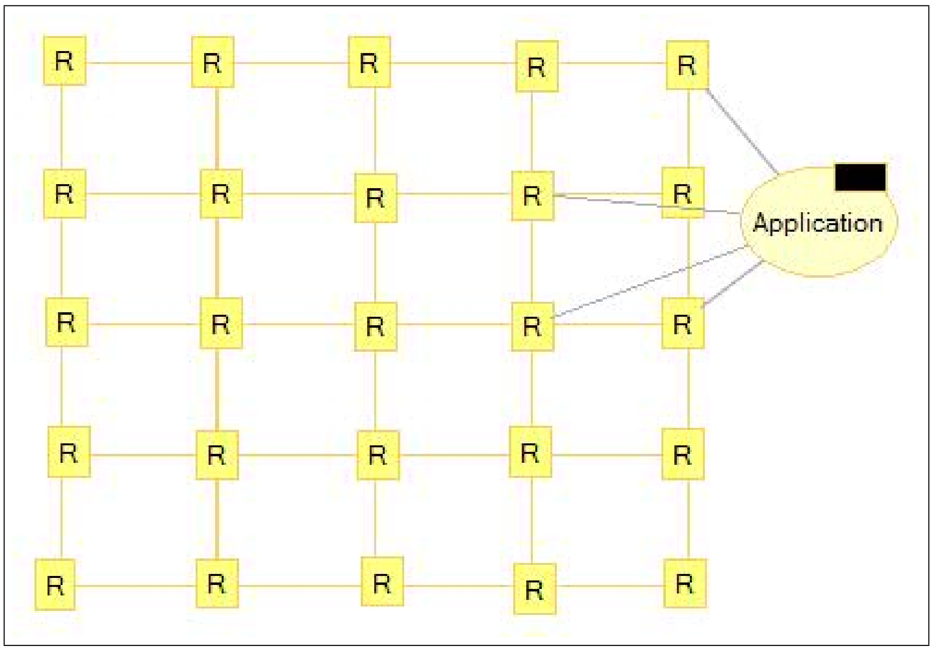


Figure 2.3: Grid network. The yellow rectangles represent resources on the grid.

composing a set of grid services.

Description A grid tries to solve the problem of coordinated resource sharing and problem solving [24]. It is more tightly coupled and hardware-focused than a service oriented architecture.This also makes it more suitable for high performance applications [24]. [24] defines a grid as an agglomeration of diverse resources in dynamic virtual organizations, while [25] states that the essence of the grid is in efficient and optimal utilization of a wide range of heterogeneous, loosely coupled resources. [25] also say that grids tie together resources to form a single virtual computer. [26] states that a computational grid consists of a set of resources, such as computers, networks, on-line instruments, data servers or sensors that are tied together by a set of common services which allow the users of the resources to view the collection as a seamless computing/information environment. These definitions have some differences, but they all agree on the notion of a grid containing interconnected distributed stateful resources.
The main technology for implementing applications on grids has been OGSI [27], created by the Globus organization. [27] defined some interfaces, grid services, that any resource on the grid had to implement. These interfaces gave access to such functionality as life cycle management and renewable references. The concepts of this standard has been refactored into Web Service Resource Framework, WSRF [23], as a response [28] to criticism from the web service community. [23] is coherent with the current web service stack of standards, such as WSDL. This standard focuses heavily on the grid as a platform for stateful resources, as opposed to services which either are stateless, or has some state associated with

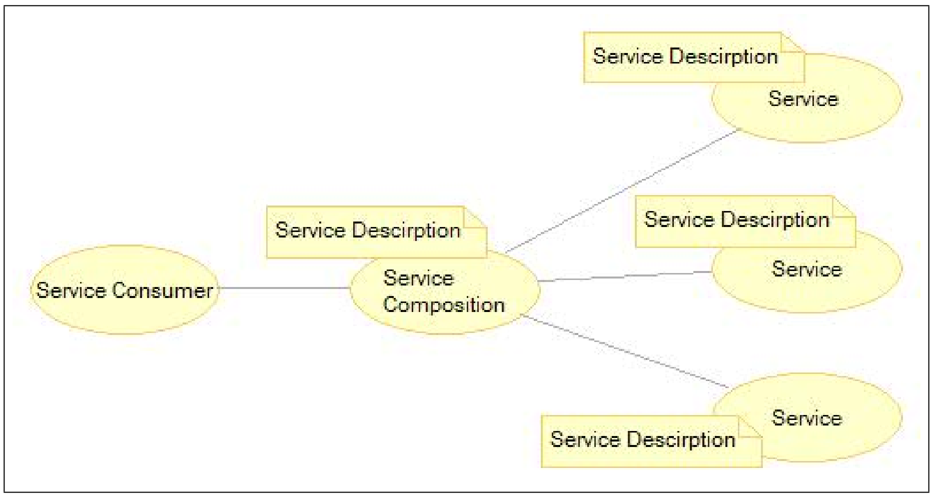


Figure 2.4: Service Composition

it defined as a conversation pattern of messages. One can build homogeneous applications on top of grids. Such applications can be created using a gridflow-language [29], [30], [31] or extending BPEL [32].
Figure 2.3 shows an architectural model of a grid. The resources are interconnected by the grid. These resources can be accesses individually, and used in a service composition, or an application can be created that use these resources, this application can then provide a service.

2.1.1.4 Service Composition

Definition A **service composition** is a service implemented by combining the functionality of other services as a network of interacting services. A service composition defines the behaviour of a composite service.

Description [15]defines a service composition as a network of interacting services, while [4] says that service composition represents usage of a set of services to accomplish a particular task. [4] also mentions the fact there is a strong tendency to use executable business processes to implement service compositions. [33] distinguishes between orchestration and choreography of services, where the former describes how services interact on a message level, and the process is always controlled by one of the partners, while the latter is more collaborative and each party describes their part in the process. A service composition is a specialized form of service orchestration where the orchestration has a external interface that is programmatically invokable.
Figure 2.1 shows the actors in a service composition. The service composition provides a service to the service consumer. The service composition combines the functionality of other services to create a composite service.

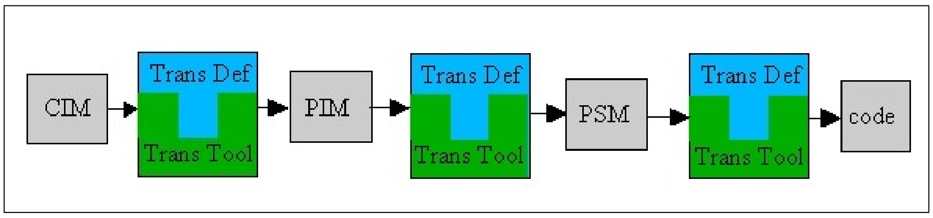


Figure 2.5: The major steps in the MDA process.

2.1.2 Model Driven Architecture

Model Driven Architecture [5] is an initiative from the Object Management Group to create a framework for software development that focuses on modelling the system, by using models with enough formalism to be understood by computers. In MDA models are created in three core abstraction layers [34]:

- Computational Independent Model (CIM) - A model that is independent of the structure of the system.
- Platform Independent Model (PIM) - A model of the system that is independent of any implementation technology.
- Platform Specific Model (PSM) - A model that is tailored to specify the system in constructs that are available in a specific implementation technology.
- Code - The executable code

To go from one abstraction layer to the next, transformations must be defined, for instance so that a platform independent model can be used as input, and the output is a platform specific model. Each supported execution platform must have its own transformation.These transformations are defined in a transformation definition language, such as QVT [35], and executed by a transformation tool, such as UMT [36]. Figure 2.5 from [34], illustrates this. The fact that a PIM can be transformed to a number of different PSM's dependent on the available transformations, makes it easier to change the implementation platform, making the development process much more portable with respect to implementation platforms.
The key benefits of using the MDA approach according to [34] are:

- Productivity. By using MDA the developer focuses more on creating the PIM thus focusing more on the domain specific problems, and less on the technical problems. This does of course assume that a transformation definition exists to transform the PIM to an executable PSM.
- Portability. The separation between PIM and PSM models aids the portability of the system. Everything that can be specified at the platform independent level is portable to any other platform, as long as there are available transformations to the relevant platform from the PIM model.

- Interoperability. If a PIM model has transformation definitions to two different platforms, is is also possible to create transformations between the two platforms. This relation between the models which are tailored for its specific platform, makes it possible to transform concepts, such as interfaces or messages, from one platform to another increasing the interoperability between the platforms.

- Maintenance and Documentation. Using a PIM model to specify the system, also creates an system documentation at an higher abstraction layer than the code. With transformation tools that are able to keep PIM and PSM models consistent throughout the development process the documentation will always be accurate.

Transformations The transformation is the process of generating a target model from a source model. There is work in progress for standardizing transformation technology for MDA, called Query, View, Transformation [35].This will be used for defining transformations as well as querying a model and creating views. [34] has identified several requirements for a transformation in MDA:

- Tunability. This gives the user of the transformation some control. The tunability can be in form of parameters or conditions in the transformation.

- Traceability. If PIM model, or a transformation is not complete, the user must complete the PSM model. Traceability means that the transformation tool should be able to trace any changes in the PSM that affects the PIM model, and update the PIM model accordingly.

- Incremental Consistency. If a user details the PSM model, after the transformation, and performs a new transformation from PIM to PSM, because changes has occurred in the PIM model, the extra detailing done in the PSM must not be overwritten by the transformation.

- Bidirectionality. Transformation should work both ways. According to [34] this is the least important of the listed requirements for a transformation.

Transformation between languages can be defined by mapping the metamodels of the languages to each other. Figure 2.6 illustrates the how model to model transformation can be used to enhance the interoperability and portability of systems. The PIM model has transformations to two different PSM models, but there are also transformations from one PSM model to another. Concepts in one PSM model can be transformed to similar concepts in the other.

Metamodelling MDA needs well defined languages, which are languages that are suitable for automated interpretation by a computer [34]. One way to define such languages is by creating a metamodel for the language. Specifically visual languages are hard to define without using metamodels. This leads to a recursive problem, because to define a metamodel, one needs a language suitable for

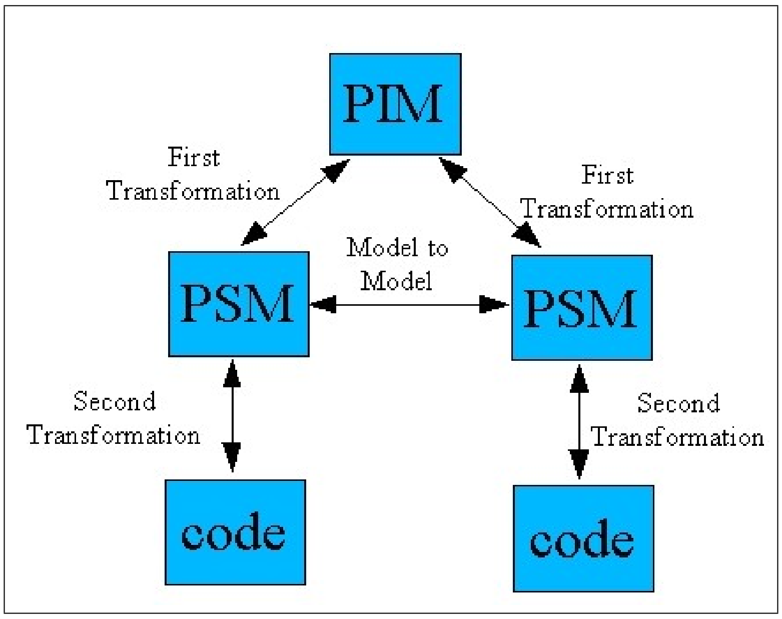


Figure 2.6: Model to Model transformation in MDA..

this, and this language is also defined by its metamodel. [34] presents the four modelling layers that OMG proposes to use for defining modelling languages.

- M0. This layer contains the running system, or the "real" instance.
- M1. This layer is where the model of the system is. An UML model would be in this layer. Objects in M0 are instances of elements in layer M1.
- M2. This layer is contains the model of the model. The metamodel of UML would be in this layer. Elements is M1 are instances of elements in this layer.
- M3. To stop the recursion OMG has defined M3 to be to top level in the hierarchy. The M3 layer is used to reason about the concepts used to define metamodels in level M2. OMG has a standard M3 language MOF (Meta Object Facility) [37].

2.2 Overview of selected visual languages

Service composition is normally implemented as a specialized form of a executable business processes. This state of the art study will therefor focus on visual languages which can be used to model business processes, but there will be specific focus on these languages' ability to support service composition and heterogeneous service technologies. This section will give an introductory presentation of the languages and a more detailed study can be found in appendix A.

2.2.1 Unified Modelling Language 2.0

UML2 [10] is a visual modelling language which contains several model views, both for structural and behavioural aspects. It is standardized by the OMG, and has standards for a lexical representation in XMI and metamodelling in MOF. UML2 is quite open and can be used to model almost any kind of system at several different abstraction layers. UML2 can be extended by using UML2 profiles. A profile specializes the constructs in UML2 to add additional meaning to them. UML2 supports modelling of business processes in activity diagrams. There is strong support for metamodelling and lexical representation of models, using MOF[37] and XMI[38]. Several tools exists for creating UML2 models.
Section A.1.1 contains a more detailed overview of UML2.

2.2.2 UML 1.4 Profile for Development for Component Based Enterprise

UML4EDOC [13] is an UML profile using UML 1.4 as the base. It uses several of the model views in UML to model component based enterprise system. The profile also has a separate view for modelling business processes. Both structural and behavioural model views are used to model a complete model of a component based system.
Section A.1.2 contains a more detailed overview of UML4EDOC.

2.2.3 Ace-GIS

[39] presents a UML1.4 profile for Web Service composition. This profile focuses on the internal aspect of the service composition with activity diagram as the model-view. The profile enhances activity diagrams with several stereotypes and tagged values. The activities use the «WebServiceCall» stereotypes to indicate an activity which is a service invocation. Such an activity uses tagged values to show the rest of the details that are needed to invoke the service, such as address information. A dataflow can also be stereotyped to indicate a transformation. A separate model view is used to model WSDL documents, based on class diagrams

2.2.4 Business Process Modelling Notation

BPMN [11] is a visual notation which is specially designed for modelling business processes. It is standardized by BPMI.org. The notation focuses on the behavioural aspects and does not have a separate view for structural models, it does however have support for differentiating between internal and external aspects of the behaviour. It has support for Web Services, but does not support other service-technologies. The BPDM [40] metamodel specification working draft from OMG contains a mapping to BPMN. There is no standardized metamodel or lexical representation for BPMN at the present. There are several tools available for creating BPMN models.
Section A.1.3 contains a more detailed overview of BPMN.

2.2.5 JOpera

JOpera [41] is developed by Eldgennossische Technische Hochschule in Zurich, and the visual composition language is a part of the Jopera service composition suite. It focuses in internal behaviour, and does have specific support for several service technologies such as support for Web Services and OGSI[27] based grid services . The visual notation is tightly integrated with the composition suite and the execution language. There is no standardized metamodel or lexical representation for this language. Section A.1.4 contains a more detailed overview of JOpera.

2.2.6 Business Processes Definition Metamodel Notation

As a part of the specification of the Business Process Definition Metamodel [40] a visual notation for business processes is used in the examples. The visual notation is further documented in a separate paper [12]. This notation builds on concepts defined in the BPDMetamodel. It contains several specialized views for both internal and external aspects of a business process and for behavioural and structural aspects. It is not based on any standardized modelling language and is not supported by any modelling tool or lexical representation. Section A.1.5 contains more detailed overview of this notation.

2.3 Related Work

Other concepts and technologies are important for a service composition. This section will focus on service implementation and description technologies and some more conceptual standards such as a metamodel for business processes and a theoretical approach to communicating processes.

2.3.1 Languages

Simple Object Access Protocol SOAP [6] is an language for invoking services. It is XML based and is used as the transport protocol for web service invocation. A SOAP message has both a header and a body. The header contains information of the operations to invoke, and other information such as address-sinformation [42], which can define where to send response messages. The body of a SOAP message contains the payload, and are sent to the service as parameters. There are also mechanics for handling exceptional behaviour, using SOAP-faults which are sent as a response to the service invocation when an error has occurred. Attachments are also possible, which makes it possible to send binary data together with the XML-based SOAP messages.

Web Service Description Language WSDL [7] is an XML based language for describing services. There is a clear separation of abstract and concrete elements. The abstract service description has defines operation signatures with input and output messages, and port-types which are collection of operations. These porttypes can be bound to concrete service implmentations in the service element. This binds the messages to datatypes defined in an xml-schema and to a protocol for transport over the network, normally SOAP[6], and the port-types to endpoint addresses. A WSDL description of a service should hold all information that is necessary for invocation of that service.

Business Process Execution Language BPEL [9] is a lexical executable language for business processes. It can be used both for both abstract and concrete process definitions. BPEL uses Web Services as the service invocation technology, and does not support any other such technologies. BPEL support both behavioural and structural modelling with constructs such as PartnerLinks available to define partner requirements.

External View Externally the interface of a BPEL process is described using WSDL, with several operations for one process. Other aspects that are externally visible are PartnerLinkTypes. This is a definition of a pair of roles that collaborate in a message exchange. The definition of these roles gives a set of requirements, given as WSDL-porttypes, for any system that want to consume the service, by implementing the specific role.

Internal View Internally a BPEL business process consists of a graph of message related activities. The activities can be invoking a service, receiving a message, or replying to an invocation. Control flow can be specified using linked activities or flow constructs such as sequence and while-loops. There is no traditional dataflow in BPEL, however the processes have variables that are available to all activities. These variables are datatypes described using xml-schema, and there are separate constructs in BPEL to manipulate these variables, using the XPath[43]. The internal part of the BPEL-process are connected with the external parts by using the partnerlinks. The invocations of a service is done by using the partnerlinks, these partnerlinks can be bound to concrete services at deployment-time which increases the decoupling of the application.

Web Service Management Layer WSML [44] uses an aspect oriented approach to introduce a separate layer between the client application and the set of web services. This layer introduces features such as dynamic selection and integration of services, client side management, and support for rules for selecting services. The WSML framework contains modules for selection, monitoring, traffic optimization, security, transaction management and billing.

Web Service Resource Framework WSRF [45] is a framework for creating applications that can execute on grids. WSRF focuses on adding service oriented capabilities to stateful resources. It accomplishes this by defining a service interface for a resource and applying a pattern for handling life cycle management, such as creating and destroying instances of the resource. WSRF is a refactoring of the concepts in the OGSI [27] framework, to make the concepts more compatible with existing web service standards. WSRF consists of a a set of specifications: Basefault [46], Resourcegroup [47], Resource Properties [48], Resource life cycle management [49] and Web Service notification [50]. The two most relevant to this thesis is briefly presented

Web Service Resource Properties [48] specifies the the definition of properties in a WS-Resource. These properties represent a view of a resource's state. A standard for messages that should be used to query or update these properties are also specified. The specification uses the example of a hard disk, that has several properties such as numberofblocks and storagecapability. These are represented as properties, while the WS-Resource also has operation such as start and stop. The properties are defined in the WSDL document in the PortType.

Web Service Resource Lifetime The Web Service Resource Lifetime specification [49] specifies a set of message interchanges that can create or destroy a WS-Resource and what WS-properties that should be used to define information that should represent the life-cycle information of a resource.

Open Grid Services Infrastructure OGSI [27] defines a Grid Service to be a Web Services that conforms to a set of conventions. OGSI uses stateful web services as the building block for creating applications running on grids. An extension of WSDL [7] is proposed, that introduced inheritance of interfaces. This extension is called GWSDL[27]. This was criticized for not being compatible with the WSDL standard. OGSI also introduces service properties, which are stateful properties that describe a service, as well as mechanism for creating instances of services and managing and accessing these instances.

JXTA The JXTA Project [20] was started by SUN, and is a set of open generalized protocols for for connections and communication between peers on a network. JXTA provides a set of platform independent basic services for the implementation of a P2P service. The specification describes several separate protocols. A JXTA network consists of several peers that can organize them selves into peer-groups to provide a common set of services. A peer advertise their services in advertisements. Communications between peers use pipes, which are asynchronous and and unidirectional. These messages are XML documents. In JXTA a peer network has several levels, where each peer group has some rendezvous peers that connect to the rendezvous peers of other groups. This makes the network more scalable as most peers only communicate directly with peers in its own group, and send messages to other groups via the rendezvous peers.

Discovery and Advertisements There are several protocols for handling discovery and advertisements of peers and its services. The Peer Discovery Protocol is used to discover new peers and their capabilities, such as services pipes. New peers can be discovered by using multicasting, or by using the rendezvous peers. Status information of a peers is discovered by using the Peer Information Protocol.

Communication The Pipe Binding Protocol is used for creating communication channels between peers. A pipe is a connection between to pipe endpoints. To find a communication routes between peers that are not directly connected, the Endpoint Routing Protocol can be used. The final protocol is the Rendezvous protocol which makes it possible for a peer to subscribe and unsubscribe to a propagation service. This protocol is also used by the other protocols to send messages.The messages can be of both java-objects and xml-documents.

Edutella Edutella[21] is a metadata structure for p2p applications. Edutella uses the JXTA[20] framework for implementing a metadata structure to use for queries in a p2p network. Edutellas vision is to enable interoperability between heterogeneous JXTA-based p2p networks.
The edutella framework consists of several services:

- Query Service. This is a standard way of representing queries and retrievals of metadata.
- Replication Service. Provides data persistence and workload balancing.
- Mapping Service. Providing translation between different metadata vocabularies. This enables interoperability between peers.
- Meditation Service. Create views that join separate metadata sources.
- Annotation Service. Annotates material stored in an Edutella network.

Pi-Calculus Pi-calculus [51] is about modelling concurrent communicating systems, as mobile, message-exchanging processes. The two main concepts of pi-calculus are processes and channels, where the channels are the communicating devices between the processes. A process has states, and action which initiates state changes. Externally the process can expose interaction, that trigger an internal action, thus making it possible to query or change the state of a process for other processes. The interaction are used for synchronization between the different processes, by communicating on the given channels. This is an analogy to B2B processes communicating and synchronizing by passing events and messages to each other.

2.3.2 Semantic Services

OWL Web Ontology Language for Services OWL-S [52] is a framework for describing Web Services semantically. There are three main aspects of a service description in OWL-S. The service profile describes the service as an entity. The service model sees the service as a process, and decomposes the service description. The final aspects, the service grounding binds the service to a concrete description.

Service Profile This profile can be used both as a method for describing the services that a service provider offers and the services that a service requester needs. A third party can then be used for matching queries from a requester to find the appropriate services. There are three basic types of information in a OWL-S service profile, the organization that provides the service, the function that the service computes, more specifically the transformation is the terms of what input and output data and any precondition and effects that are defined for the service, and a set of features that specify the characteristics of the service. For the last type, a classification system must be used to describe the service characteristics, such as a Quality of Service framework.

Service Model OWL-S models services as processes, using a subclass of its ServiceModel, the ProcessModel to describe processes. The process model can be used to describe more complex service, by using the composite process construct in the model. The atomic process can be mapped to a single operation, but a composite process can contain several processes, thus making it possible to create processes that implement the conversation pattern. Several control flow constructs exists to model the external behaviour of the process such as split-join, any-order, sequence, choice and iterate. This make is it possible to describe complex behaviour of a service and its protocol.

Service Grounding This specifies the details necessary to actually invoke the service, such as protocol and message format, serialization, transport and addressing. This part of the OWL-S specification is tightly connected with WSDL and its binding concepts. While WSDL contain both the abstract and the concrete definition, the service grounding in OWL-S only contain the concrete information ,with the abstract definitions being in the process model. What OWL-S can describe that is outside the scope of WSDL is the OWL classes, which can be seen as the abstract types of messages as they are defined in WSDL.

Web Service Modelling Ontology The WSMO[53] is a conceptual model for the description of semantic Web Services. This ontology is the bases for the Web Service Modelling Language [54]. WSMO makes it possible to reuse ontologies by importing other ontologies and use them in describing a service. WSML is the formal language for description of Web Services based on WSMO.

2.3.3 Quality of Service

Web Service Quality of Service WS-QoS [55] presents a framework for enabling Quality of Service-aware Web Services. This framework uses a XML-based language for defining Quality of Service-attributes, used by both the service provider and service consumer. This attributed are added to the service description, and there is a broker which is aware of these attributes and can match the requirements of a service consumer and quality of service attributes provided by the service provider. An ontology defines the attributes and metrics for used and the associations between these.

OMG UML Profile for Quality of Service [56] specifies a UML-profile for defining Quality of Service attributes using UML. The main part of this profile is the QoSCategory. A category contains QoS characteristics. These characteristics can be associated with other characteristics by using QoSDimensions.

2.3.4 Model Driven Architecture

Query View Transformation QVT [35] is work in progress being performed by the OMG. The purpose is to defined a language or a set of languages to create a view on a model, to query a model and to writing transformation definitions. The latest proposal uses MOF 2.0 and the Object Constraints Language (OCL) from OMG as its base. Declarative programming concepts are used to defined relations between MOF based models, while an imperative language is used for defining transformations.

UML Model Transformation Tool UMT [36] is a graphical tool for executing transformation between models in a Model Driven Architecture. UMT is an XML based tool using XSLT [57] for defining transformation. UMT takes UML models represented using XMI [38], transforms this to an internal XMI dialect called XMI Light. This is used as the source for the transformation. XSLT can produce any kind of text-based output, but is best suited for XML. UMT can produce XML based output, including XMI for representing UML models and XML based execution languages such as BPEL[9].

2.3.5 Other

Business Process Definition Metamodel This specification [40] is an initiative from OMG to create a common metamodel for all languages that want to describe business processes, both visual modelling languages and lexical executable languages. The metamodel focuses on both the structural and behavioural aspects of business process with concepts such tasks, roles and resources. The current working draft of the specification proposes to use a subset of the UML metamodel and to add some stereotypes as well to introduce concepts such as resource, role and process. As can be seen from the metamodel in figure 2.7

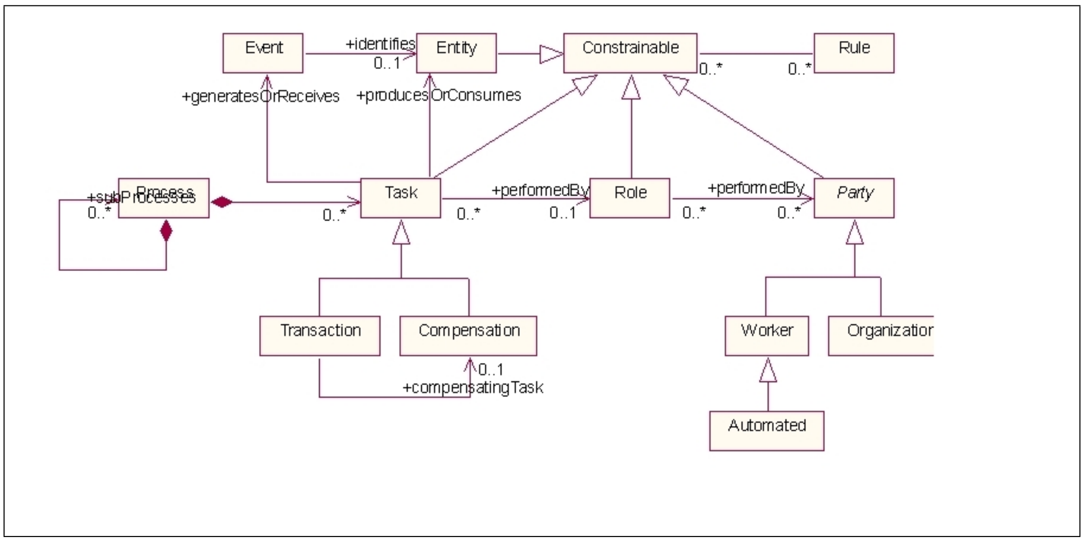


Figure 2.7: Business Process Definition Metamodel

it uses quite simple concepts. A process contains tasks, or subprocesses, and each task is performed by a role. The role is an abstract definition and must be realized by a concrete implementation, which can be either a software component or organizational unit. A task can also use events for communication with other tasks.
BPDM proposes to be the metamodel for BPEL and BPMN, and the specification contains mappings from the BPDM concepts to both these languages.

Chapter 3

Requirements

This section will present the identified requirements for a platform independent visual language for modelling heterogeneous service composition. Firstly requirements for a service composition language from an external viewpoint will be defined. The next section presents the requirements for modelling service composition from an internal viewpoint. In the second part requirements that are specific for the relevant service implementation technologies (Web Services, Grid Services, P2P network Services) will be presented.

3.1 External Service Composition Requirements

This section discusses the requirements for a service composition that is related to what is externally visible, both structural and behavioural. This includes the service description definitions as well as the how the composite service relates to its consumer and the composed services.

3.1.1 Service Description

- The modelling language should support models describing the service composition from an external point of view, detailed enough to be used by a service consumer for automatic service invocation.

The main aspects of this is the functional information, such as operation signatures , with data and exception definitions and a protocol for a complete service interaction. The interface description describes the static aspects of the service, while the protocol describes the expected behaviour in terms of message exchanges. Concrete service information such as addresses should also be given. The importance of a service description can be seen from figure 2.1, where the service description document is used by all partners in a service interaction. A conversational service such as the one described in the RFP-case, appendix C, where several messages are exchanged during a service executions, requires a more complex service description also including protocol, which is the the order of the expected message exchanges.

3.1.2 Partner Definition

- It should be possible to model the requirements that must be fulfilled partners, both service consumers and composed services.

The partners of the service, both the service-consumers and the composite services are important aspects, thus it is important that a a visual language makes it possible to model the structure and behaviour that the composed service needs from its partners to function correctly, such as interfaces they must implement and a protocol for communication.The service composition can interact with these partners several times during an execution of a service.
BPEL [9] introduces this concept, calling it PartnerLinkTypes which is a definition of the interfaces that each partner in an interaction must provide to be able to execute the service composition. In the RFP case, appendix C, this is required as the customer starting the service must be able to receive messages from the service composition on asynchronously.

3.1.3 Instance Access Management

- It should be possible for the service consumer to control which instance of the service that receives a message.

A service with many consumers will at any given time have several running instances, each one handling interaction with one service consumer, running at the same network address. If the service is a stateless request/response this is unproblematic, as the service consumer sends a request that initiates the instance of the service, and when the response is sent back, the instance of the service is no longer needed, and the instance can handle another request. If, however, the service is asynchronous and conversational, sending several messages back and forth, the service needs a more complex set of features to handle this. This interaction pattern can also be present between the composite service and the composed services, thus making it necessary to handle the same set of challenges. When several instances of a service is running simultaneously, and using the same endpoint, there must be some mechanism for the communicating partners to uniquely identify the instance they want to communicate with. This should be as transparent as possible for the involved partners, either using some piece of data as a conversation id, or using some mechanism for detailing in the endpoint which receives the message. The modelling language should be able to model both when such features are to be used, and how they should be implemented.
This requirement is specifically important in cases involving conversations such as the RFP case, appendix C. If there are two parallel executions of this service composition with two different customers, both sending messages to the same endpoint it must be possible to direct the message from the customer to the same instance of the service composition that the customer started its conversation with.

3.2 Internal Service Composition Requirements

This section will detail the requirements for service composition that is related to the internal behavior and structure of the service, the expressiveness and what kind of communication it supports.

3.2.1 Expressiveness

- The modelling language should support at least the 5 basic workpatterns [58]: sequence, parallel split, synchronization, exclusive choice and simple merge. These workpatterns are described in detail in appendix B.

A service composition has several similarities with workflow [59], consuming a service, sending or receiving a message can be considered a piece of work, and these can be coordinated in a control-flow. The data being used by the tasks can be defined in a data flow. Several workpatterns have been identified and described by the workflow community, presented here [58, 60], which can be used as a framework for specifying the required expressiveness of a service composition language.
This work focuses on direct support for workpatterns, meaning that there is a construct in the language for handling the workpattern. For visual languages, such as UML and BPMN [61] shows that even though a language does not directly support a work pattern, one can implement these patterns by combining several constructs. See appendix B for more on this.

3.2.2 Data Manipulation

- It should be possible to send parts of a data object as a dataflow, update parts of an object and transform simple atomic data values.

Data flows into and out of tasks or activities. The data can be a complex hierarchical object, and some activities or tasks may only require a subpart of this object. In such a scenario it must be possible model manipulation of this dataobject so that subparts can be identified and extracted and used as input to another activity or task. Collections are an example of this were an activity uses one item in a collection at the time for execution of the activity.
Transformation of basic data values, such as strings and integers, should be possible to model in a service compositions language. Examples of this would be counters in loops or concatenating tow strings.
The RFP case, see appendix C, shows examples of this behaviour. The RFP object that is sent from the customer to the service composition contains a "category" member. This data must be extracted from the RFP object before it is sent to the SupplierRegistry service. Another example is from the Distributed Office Backup case, see appendix E, where an collection of peers are returned from the search of the p2p network. The item in this collection is processes individually in the next activity.

3.2.3 Communication

- It should be possible to model synchronous service invocations..
- It should be possible to model asynchronous message-based communication.

Communication in the context of an internal activity graph, is what types of communication paradigms are supported by the tasks or activities. Synchronous invocations as well as asynchronous send and receive should at least be supported, but for some special cases, streams, signals, and multicast could be useful.

Synchronous Synchronous communication request/response, where the requester blocks its processing until receiving a response, is best suited for the basic stateless services. This maps equal to invoking an operation on the service and the requirements the service consumer has to meet are basic, such as supporting the same transport protocol as the service. The modelling language should have constructs for invoking such a service, defining the input and output data.
This service invocation paradigm is supported by BPEL [9] and an example can be seen in the RFP case, appendix C, where the interaction between the RFP-service and the SupplierRegistry is synchronous.

Asynchronous Asynchronous communication increases the independence between the partners in the service composition, creating a more decoupled architecture. Asynchronous messaging is often implemented using a callback interface, a reference to an endpoint where the composite service can send the response back to the service consumer. To successfully implement this, the service composition language must be able to dynamically set the endpoints that should be used, from information received at runtime. Such an implementation means that all the partners involved in the service execution must fulfill the requirements of the defined roles they implement in the conversation.
This communication paradigm is defined in one of the WSRF specifications, WS-Notification [50], which supports asynchronous messages between services using a pattern of web services.

3.2.4 Conversational Services

- It should be possible for the service composition to interact with other service in a conversational manner, with several message exchanges between service instances.

A conversational service is a service that performs a set of interaction between partners in one service execution. An activity representing an message exchange with a partner must contain information about which partner should be used in the interaction, as the service composition instance can have several conversations

with different partners at the same time.
BPEL[9] uses PartnerLinks as instances of PartnerLinkTypes to indicate which partner-instance is used in for each interaction. The RFP-case, appendix C, has several examples of this scenario, where the service composition at the same time has several running conversations with different partners.

3.2.5 Dynamic Service Selection

- It should be possible to abstractly define partner services, and bind these dynamically to concrete service instances.

To increase the decoupling between the service composition and the composite service implementation dynamicy can be introduced for service selection. This could be used for adding fail-over solutions or just a part of the business logic, where one service is chosen over another based on some dynamic quality criteria. UDDI or similar service brokers is a possible solution to this, where instead of defining a service instance, a service broker instance and a query for matching services, as well as a policy for handling multiple results to the query, is defined. To handle this the service composition language must support using query and policy for identifying a service rather than a address based-reference to a service in a transparent way. The distributed office backup case, appendix E, has this requirement, where the backup device being used is dynamically decided at runtime.

3.3 Other Requirements

3.3.1 Modularity

- The modelling language should be modular to support reuse of models.

Visual models is modularized by using several diagrams, interconnected by model-elements such as ports. By using these elements one can create an hierarchy of models, increasing the detailing as one traverses down the hierarchy. Different aspects of the models can be modelled in different types of models, such as the structure of the data, and how the data flows in the composed service. To make this possible one can create modules of parts of the service definition, so that this part can be used by several other services. Sometimes it is more appropriate to extract this functionality to a separate service, which can be consumed by others.

3.3.2 Consistency

- All model views in a model should be consistent.

When a model has utilises several model views, and a set of models is used together to model a system, consistency is a main factor. An object in the model should only have one definitoin to avoid ambiguity. This is particularly important if a model is the source of a transformation, which increases the importance of a valid and consistent model. The best way to ensure consistency is to have a clear separation of concern in the structure of the model, with definition of each model element only once.

3.4 Heterogeneous Services

3.4.1 Web Service Composition Requirements

- It should be possible to model service consumption and service description using Web Service technologies, such as SOAP and WSDL.

Web Services are a set of standards for implementing a service oriented architecture, thus the main requirements that come from basing the service composition on web services are supporting the web service standards. SOAP [6] for messages, WSDL [7] interface and service description and also WSA [42] for endpoint addressing. Other standards that could be supported are the WS-Transaction [17] and WS-Security [16] as well as the service broker functionality of UDDI [8].

3.4.2 Grid Service Composition Requirements

- It should be possible to model composition of stateful resources implemented on grids in the service composition.
- It should be possible to model life cycle management of a stateful resource.
- It should be possible to model accessing of the properties of a resource.

A grid consists of a set of interconnected resources. These resources can be stateful, thus have requirements with regards to life cycle management by the client, and have and externally visible state in the form of properties. WSRF [45] proposes a service oriented paradigm for using and accessing resources on a grid. WSRF proposes patterns for stateless services for life cycle management and accessing properties.
Some ambiguity exists in the term Grid Service. This can be the service provided by a single resource on a grid, or a service which is implemented by combing the services of several resources. The latter case is likely to be a stateless service providing some function as a regular stateless service, thus introducing no extra complexity or requirements, while a single resource create the extra complexity already mentioned.
The disk-drive case, see appendix E, has an example of consuming the service provided by a single stateful resource on the grid. The disk-drive has properties describing its state, as well as operations for life cycle management, start and stop operations.

3.4.3 P2P Network Composition Requirements

- It should be possible to model composition of services implemented by peers in a p2p network in the service composition.

In the case of a peer-to-peer service being the service provided by a a single peer, in a peer-to-peer network, the service consumer must be another peer on the network, and a search for peers providing the required service must be performed before consuming the service.
A service implemented by a peer-to-peer network, will not have this requirement, as all the complexity of peer-to-peer communication is hidden inside the service, and the service can be provided to the service consumer as a regular stateless service.

3.5 Summary of requirements

Tables 3.1 and 3.2 summarizes the requirements presented in this chapter. These requirements will be used for several evaluations throughout this thesis.

	Requirement	**Detail**
External	Service Description	The modelling language should support models describing the service composition from an external point of view, detailed enough to be used by a service consumer for automatic service invocation
	Partner Definition	It should be possible to model the requirements that must be fulfilled partners, both service consumers and composed services.
	Instance Access Management	It should be possible for the service consumer to control which instance of the service that receives a message.
Internal	Expressiveness	The modelling language should support at least the 5 basic workpatterns [58]: sequence, parallel split, synchronization, exclusive choice and simple merge.
	Data Manipulation	It should be possible to send parts of a data object as a dataflow, update parts of an object and transform simple atomic data values.
	Communication	It should be possible to model synchronous service invocations
	Communication	It should be possible to model asynchronous message-based communication.

Table 3.1: Summary of all requirements presented in this chapter. This table continues on the next page

	Requirement	**Detail**
	Conversational Services	It should be possible for the service composition to interact with other service in a conversational manner, with several message exchanges between service instances.
	Dynamic service selection	It should be possible to abstractly define partner services, and bind these dynamically to concrete service instances.
Other Requirements	Modularity	The modelling language should be modular to support reuse of models.
	Consistency	All model views in a model should be consistent.
Web Services	Web Services Standards	It should be possible to model service consumption and service description using Web Service technologies, such as SOAP and WSDL.
Grid	Grid Services	It should be possible to model composition of stateful resources implemented on grids in the service composition.
	Resource life cycle	It should be possible to model life cycle management of a stateful resource.
	Resource properties	It should be possible to model accessing of the properties of a resource.
P2P	P2P Network Services	It should be possible to model composition of services implemented by peers in a p2p network in the service composition

Table 3.2: Summary of all requirements presented in this chapter (continued)

Chapter 4

Case descriptions

This chapter presents the cases used in this thesis. The cases are implemented using existing standards UML2 and BPMN, and AuSCL. The evaluation of the modelling languages are based on the case implementations. UML2 and BPMN are evaluated in chapter 5, while AuSCL is evaluated in chapter 7.

4.1 Case A - Request For Proposal

This is a case based on conversation between several partners. The goal of the service is to get proposals from several suppliers based on a "Request for Proposal" message from the customer. A use case diagram of the case can be seen in figure 4.1

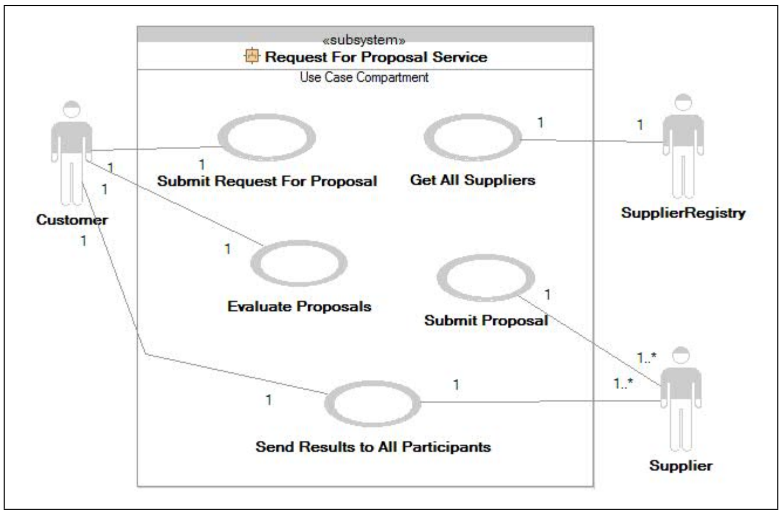


Figure 4.1: UML2 Use case description of the RFP case.

Actors

- Customer - The sender of the Request For Proposal (RFP). The customer must be able to receive messages from the RFP service as well as send messages.
- RFPService - The service which orchestrates the consumption of the other services. This actor orchestrates the service interactions with all other actors.
- SupplierRegistry - A yellow-pages type service, with one service returning all suppliers which are available in the given category.
- Supplier - A supplier that can perform the work described in the request for proposal. This actor must receive messages from the RFPService and also send messages back.

Behaviour First a Request For Proposal-message is sent from the customer to the RFPService. Based on the contents of this message, the RFPService sends a query to the SupplierRegistry to get information of all appropriate suppliers. When the RFPService gets this list, the RFP is sent to all suppliers. The supplier sends a proposal back to the RFPService, which collects all proposals in a list, and sends the list to the customer. The customer sends back its decision of which proposal is the best. The RFP service then sends this result to all suppliers that submitted a proposal, and send information about the winning supplier to the customer.

Issues

- Conversations - There are conversational interactions between the Customer and the RFPService and the RFPService and the suppliers.
- Asynchronous Communication - The communication between the RFP service and its partner is asynchronous.
- Partner-requirements. To consumer the service the Customer must provide some services as well. These services are used for communication between the RFP-service and the customer during the execution of the service.
- Dynamic service selection. The RFPService does not know the details of the suppliers before it is received from the SupplierRegistry. RFPService must handle a list of service-endpoint definitions and at runtime use this information to invoke the services.

4.2 Case B - Generic Disk Drive

This case is described in one of the WSRF subspecifications [48] as an example. A disk drive is the stateful resource on a grid which can be accessed as a WS-Resource through a Web Service interface. This WSRF enabled example is different from a pure Web Service example as it contains life cycle management of stateful resources, in this case by starting and stopping the diskdrive. A use case diagram of the case can be seen in figure 4.2

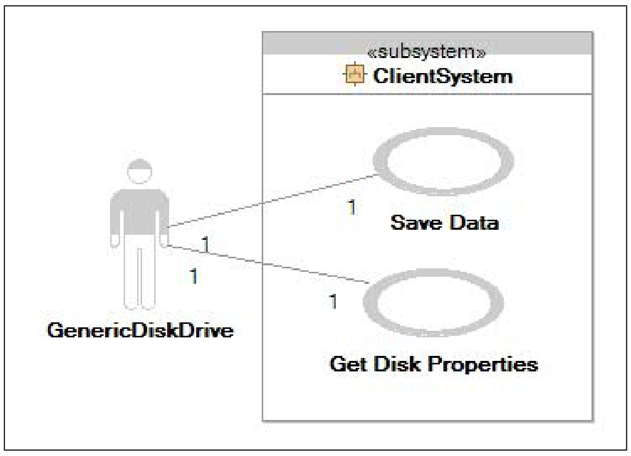


Figure 4.2: UML2 Use case description of the Generic disk drive case.

Actors

- DiskDrive - This disk drive contains some resources that describe its state. There are also operations for starting, stopping and storing data on the drive. The drive must be started before data can be stored on it.
- Client - This is the client that wants to store data on the disk.

Behaviour The client starts the disk, gets the properties of the disk to see that these properties indicate the the disk is able to handle storing the relevant file. The the client store the data on the disk, before stopping the disk.

Issues

- Life cycle - The disk drive has a predefined life cycle, and must be started by an operation before it is used for storing data.
- Properties - The resource provides properties to clients as well as operations.

4.3 Case C - Distributed Office Backup

This case is based on a service that backs up a files from a distributed peer-to-peer office collaboration application to a tape-disk, with some options available for backing up only the newest found version or all found versions of some resource. The office environment is implemented in a P2Protocol such as JXTA and have services for looking up file resources on the network, and the tape-disk is a stateful resource on a grid, with properties describing the amount of available space on the disk, and operations for starting, stopping and storing. This service composition is made available as a web service, so that other applications can use it to get back up of files. A use case diagram of the case can be seen in figure 4.3

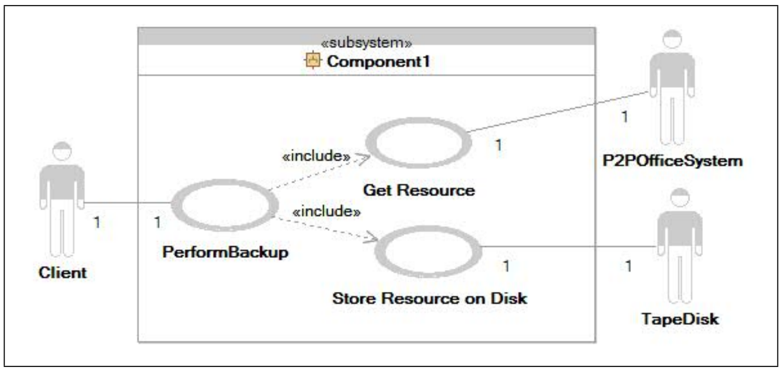


Figure 4.3: UML2 Use case description of the Distributed Office Backup.

Actors

- Client - The client want to have backup taken of a file-resource.
- BackupService - The backupservice receives the backup-request and finds the file and stores it on a tape disk. The Backupservice must act as an peer in the P2P network. The service is exposed as a Web Service.
- P2P Office Collaboration Network - This is a P2P based office collaboration application which makes it possible to have a collaborative environment without a central file server. Each peer in the network has the ability to search for file-resources on all the other peers through a search mechanism.
- Tape disk - The tape disk is a backup device available on a grid infrastructure. This disk contains properties that describe its state, with information such as available diskspace. The tape disk must be started before properties can be retrieved or data can be stored. Which disk to use should be decided at runtime.

Behaviour The Client sends a backup-request to the backupservice, containing the name of the file-resource that should be backed up. The BackupService searches for this on the P2P network, and from this gets a list of peers with this resource. If the backup should be of the newest version only, this is downloaded, otherwise all found versions are downloaded. The tape disk is started and the available space is checked against the size of the downloaded resource. If there are enough space available on the disk, the resource is stored.

Issues

- P2P network service invocation. The BackupService must invoke a service on a P2P network, searching for the peers providing this service before invoking the service on the peer.
- Heterogeneous Services - P2P networks, stateful resources on a grid, and Web Services are all used in this case.

4.4 Relationship between requirements and cases.

Table 4.1 provides a mapping of these requirements to the case implementations used in this thesis. All case implementation models can be found in the appendices of this thesis.

- Case A - Request for Proposal, see appendix C for all models. This case is implemented in UML2, BPMN and AuSCL
- Case B - Generic Disk Drive, see appendix D for all models. This case is implemented in UML2 and AuSCL.
- Case C - Distributed Office Backup, see appendix E for all models. This case is implemented in UML2 and AuSCL.

	Requirements	**Case A**	**Case B**	**Case C**
External	Service Description	*	*	*
	Partner Definition	*	*	*
	Instance Access Management	*		
Internal	Expressiveness	*	*	*
	Data Manipulation	*		
	Communication - Synchronous	*		
	Communication - Asynchronous	*	*	*
	Conversational Services	*		
	Dynamic Service Selection	*		*
Other	Modularity	*	*	*
	Consistency	*	*	*
WS	Web Services	*	*	*
Grid	Access a stateful resource		*	*
	Lifecycle management		*	*
	Access Properties		*	*
P2P	Consumer service on a peer			*

Table 4.1: Table of relation between requirements and case studies

Chapter 5

Evaluation of Existing Solutions

This chapter presents the evaluation of two selected visual languages that can be used to model service composition. UML2 and BPMN are selected based their potential for modelling service composition and tool support. The evaluation will be based on the implementation of the presented cases. The cases are selected to test the languages' ability to model composition of heterogeneous service compositions, and will include web service, grid services and p2p-network services together with other complex aspects such as conversations. The cases are presented in chapter 4, while the relationship between the cases and the requirements are shown in table 4.1. The evaluation of the languages will be structured according to the table of requirements shown in table 5.1, and detailed in chapter 3.

	Requirements
External	Service Description
	Partner Definition
	Instance Access Management
Internal	Expressiveness
	Data Manipulation
	Communication - Synchronous
	Communication - Asynchronous
	Conversational Services
	Dynamic Service Selection
Other	Modularity
	Consistency
WS	Web Services
Grid	Access a stateful resource
	Lifecycle management
	Access Properties
P2P	Consumer service on a peer

Table 5.1: Table of requirements

5.1 UML2 - Service Composition

All three cases presented in chapter 4 have been implemented using UML2[10]. A complete set of the models, as well as case descriptions, can be found in appendices C.2, D.2, E.2. All models are created using Rational Software Modeller by IBM.

5.1.1 External Requirements

External requirements are requirements related to a service composition seen from an external viewpoint, without considering the internal details of the service composition. UML2 has several model views focusing on external models, such as sequence diagrams and component diagrams.

Service Description UML2 uses class diagrams and interfaces to model the functional aspects of a service. An interface consists of several operations, and the signature of an operation defines the input and output data for an operation in addition to an operation name. This is exemplified by figure 5.1 which is an interface from the Request for Proposal-case, described in section 4.1.

The datatypes used in an operation, as parameters and results values, can

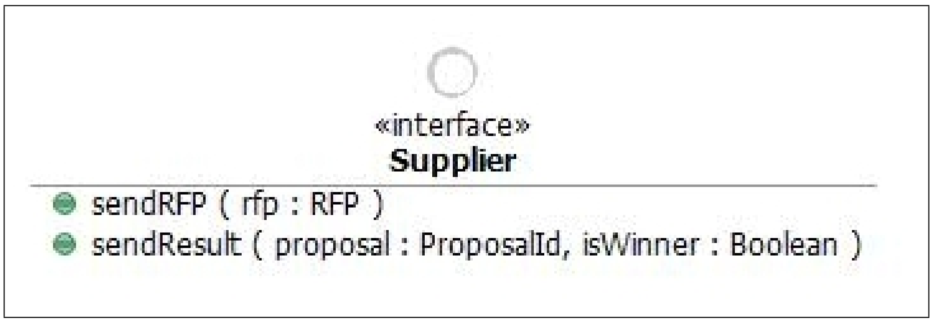


Figure 5.1: UML2 Interfaces. The Supplier Interface contains two operations. The ball notation is used to indicate that this is an interface.

be modelled using UML2 class diagrams. Operations are grouped in interfaces, while interfaces can be grouped by ports if necessary. Using UML2 component diagrams, one can model both the interfaces that a service *provides* and the interfaces which are *required* by the service. Figure 5.2 shows a set of interconnected components being connected to other components by wiring the interfaces, using the ball/socket notation. The ball represents the provided interface, and the socket represents the required interface. A dependency is introduced from the required to the provided interface. An interaction diagrams can model the protocol fro a service, specified by the expected sequence of messages exchanges between partners. A subset of the protocol of the RFP-case is shown in figure 5.3. All together these separate model views gives a complete model of the service description.

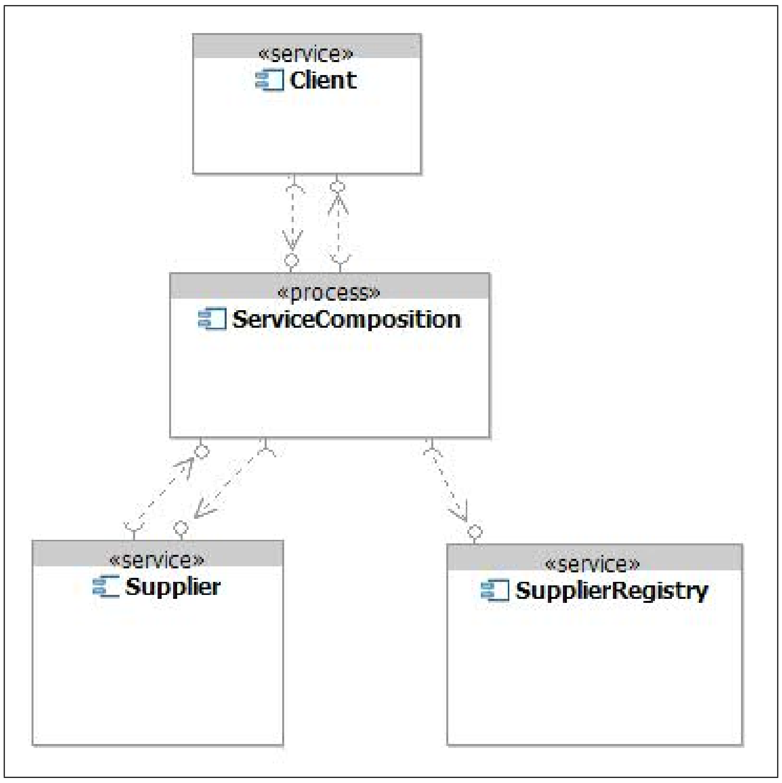


Figure 5.2: UML2 Provided and Required Interfaces. The ball/socket notation is used to show provided and required interfaces. Dependencies from the required interface to the provided interface connects component pairs.

Partner Definition By modelling UML2 components with provided and required interfaces, these model views represent requirements to any partner in a collaboration. A partner that wants to consume a service, that has some required interfaces, the partner must implement these interfaces, to handle potential message exchanges, before it can consume the service. On an abstract role-based level, a collaboration between partners can be described using collaboration diagrams, defining the roles for each participant, and specifying the behaviour for the roles in the context of that collaboration as sequence diagrams. Figure 5.2 shows how this is modelled in an UML2 component diagram.

Instance Access Management An lifeline in a sequence diagram represents an instance of a specification, and further details about these instances can added by using the execution occurrence model-element. This makes it possible to model precisely what instances are used during a message exchange. It is not possible, however to model what mechanisms to implement for this. Figure 5.3 shows an sequence diagram, where the execution occurrences represents the specific instances of the partners used in the interaction.

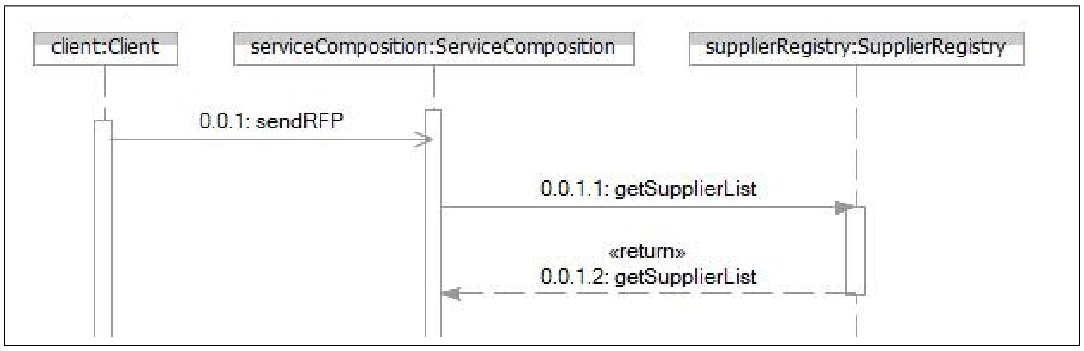


Figure 5.3: UML2 Interaction Diagram shows three partners exchanging messages. Execution occurrences is used to indicate the instances of the partners.

5.1.2 Internal Requirements

Internal behaviour of a service composition is modelled in UML2 as a activity graph. In an activity diagram, an action can be used to call an operation provided by another component. This makes it possible to model an activity graph where the actions are invocations of partner services.

Expressiveness - 5 basic workpatterns The cases used for this evaluation do not have any complex requirements with regards to expressiveness. As described in appendix B, UML2 supports most of the workpatterns found for a flow-based language [58]. The 5 basic workpatterns (sequence, parallel split, synchronization, exclusive choice and simple merge), which are used in the cases are all supported by UML2.

Data Manipulation UML2 activity diagrams do not have explicit support for the manipulation of data objects that are passed between actions. The most appropriate way of handling data manipulation is to create a set of actions for that purpose. The problem is that there is no standard way of describing the implementation of such a set of actions . One could use the UML action language, which has actions for reading, writing, and updating data elements, but this is cumbersome for larger manipulations, with complex logic. A second possibility is to use the Object Constraint Language (OCL)[36]e as a lexical alternative, but it is not well defined how OCL shall be used for this purpose. A third option as presented in figure 5.4 where a constraint is added directly to a dataflow. This is best suited for simple data manipulation such as using only parts of a data object as input to the the target action.

Communication An action can be used to model a service invocation, by using a *CallOperationAction* that invokes an operation on another component. In figure 5.5, from the RFP-case, the "getSupplierList" action is such an *CallOperationaction*, invoking an operation on the SupplierRegisty component. However,

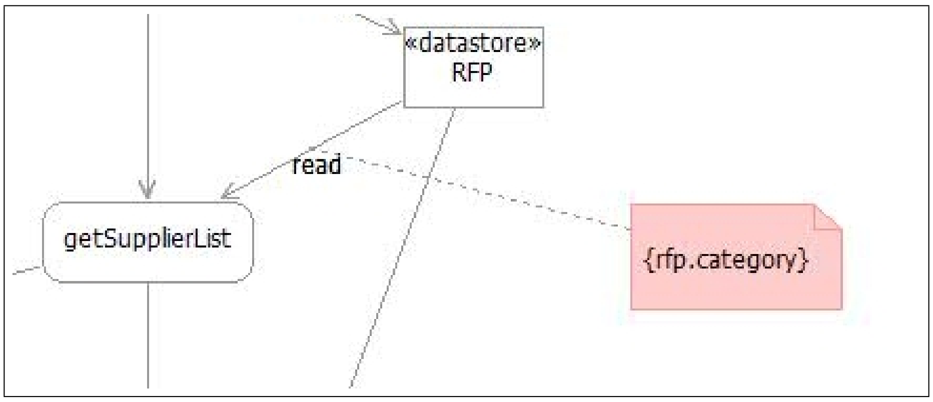


Figure 5.4: UML2 Dataflow with constraint

UML2 does not support a standard visual construct for modelling the connection between the calloperationaction and the partner providing the operation.
The "ReceieveRFPFromClient" action models receiving a message. It is not possible to model in an activity diagram what operation is used for transporting this message to the internal process in the same way as when invoking a service on an external partner.

Using activities for modelling communication with external partners has a

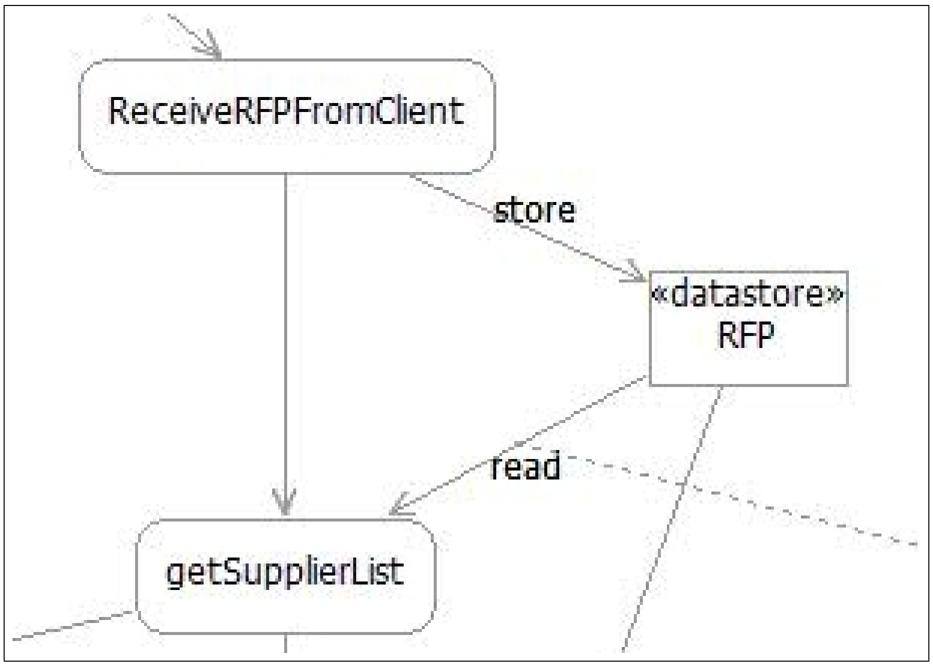


Figure 5.5: UML2 Actions for invoking services and receiving messages

weakness in that it is not possible to visually model structural aspects of the communication such as who to send the message to, what message to send and what operation to use for sending the message. This information can be modelled using a sequence diagram, but this would lead to overlapping model views, showing the same behaviour making it possible to create inconsistencies.

Conversational Services When participating in conversations, both internal and external model views should be defined, internal model views to show the behaviour representing communication with the partner, while the external model views should present the details of what partner the communication is related to. UML2 lacks some support for consistency between internal and external behaviour modelling, using activity and sequence diagrams. This makes it hard to model conversations in a consistent manner.

Dynamic Service Selection The performer of an activity can be modelled in two different ways in UML2, either by using swimlanes, where the lane is specified by an role or as a class. An activity, or action, can be linked with an operation on another component in the environment. These links to external partners is defined at design time, and there is no construct in UML2 for using properties to set these partners at run time. This can be modelled explicitly as a part of the service composition logic, using a service discovery mechanism for finding appropriate partners, but there is no direct support in UML2 for specifying such dynamic service selection

5.1.3 Other Requirements

Modularity Many of the model views in UML has excellent support for modularity. In activity diagrams an activity can be decomposed into another activity to create a hierarchical structure. In interactions one can decompose lifelines to show internal behaviour, and one can also reference other interactions to create hierarchies. Both class and component diagrams can be decomposed and detailed, by using composite structure diagrams, where the internal parts of a component or class can be a component or a class. There is however a lack of defined structure for models containing several model views. This must be introduced by the modeller.

Consistency The multiple model views of UML2 makes it possible to have definitions of the artifacts used in the model separated from the usage of the constructs, such as class diagrams defining classes and interfaces, used by other model views, for instance as messages in sequence diagrams. Behavioural aspects such as the communication and sending and receiving of messages are more problematic, as there are no direct connection between tasks modelling sending and receiving messages and sequence diagrams modelling messages going between partners. Figures 5.7 and 5.8 shows the activity diagram and sequence diagram of the same behaviour. There is no direct link between the start-activity in the activity diagram and the sending of the start-message shown in the sequence diagram. This behaviour is defined twice, making it vulnerable for inconsistency.

5.1.4 Heterogeneous Service Requirements

This section evaluates UML2's capabilities to handle composition of services using Web Service technologies, Grid based technologies or P2P based technologies.

Web Services UML2 is a contains full support for the service invocation paradigms used by Web Services. The WSDL service description model is similar to interfaces using UML2, and the grouping of interfaces can be done in UML2 by using ports. The invocation of stateless services, is modelled in UML2 by simple operation invocation.

P2P Network Services Invoking a service implemented on a P2P network is not very different from invoking a service in a regular client/server environment. The decentralized and unstable nature of such a network might mean that one should always search for a relevant peer to invoke the service on, before the invocation. The service invocation in itself is similar to invoking a regular service. An example of invoking a Web Service using UML2 can be seen in figure 5.6, where a sequence diagrams shows the message for finding a peer, then a message for invoking the service.

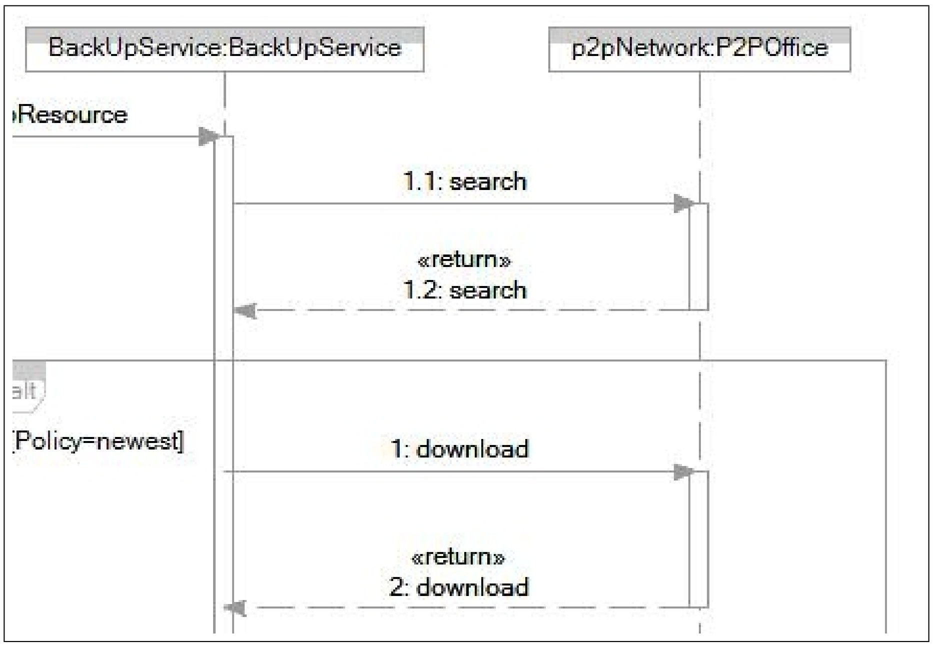


Figure 5.6: UML2 Protocol for accessing a P2P network service.

Grid Services A grid contains resources that can be stateful. Using WSRF it is possible to access such resources with a web service. To be able to model consumption of grid services one must be able to model the life cycle management that is necessary to consume such services. By using activity diagrams to model the tasks of sending and receiving messages it is not easy to model the life cycle of the resource in the collaboration, as the activity diagrams only models internal aspects of the local service. A activity diagrams showing communication with a WS-resource can be seen in figure 5.7 By using sequence diagrams this can

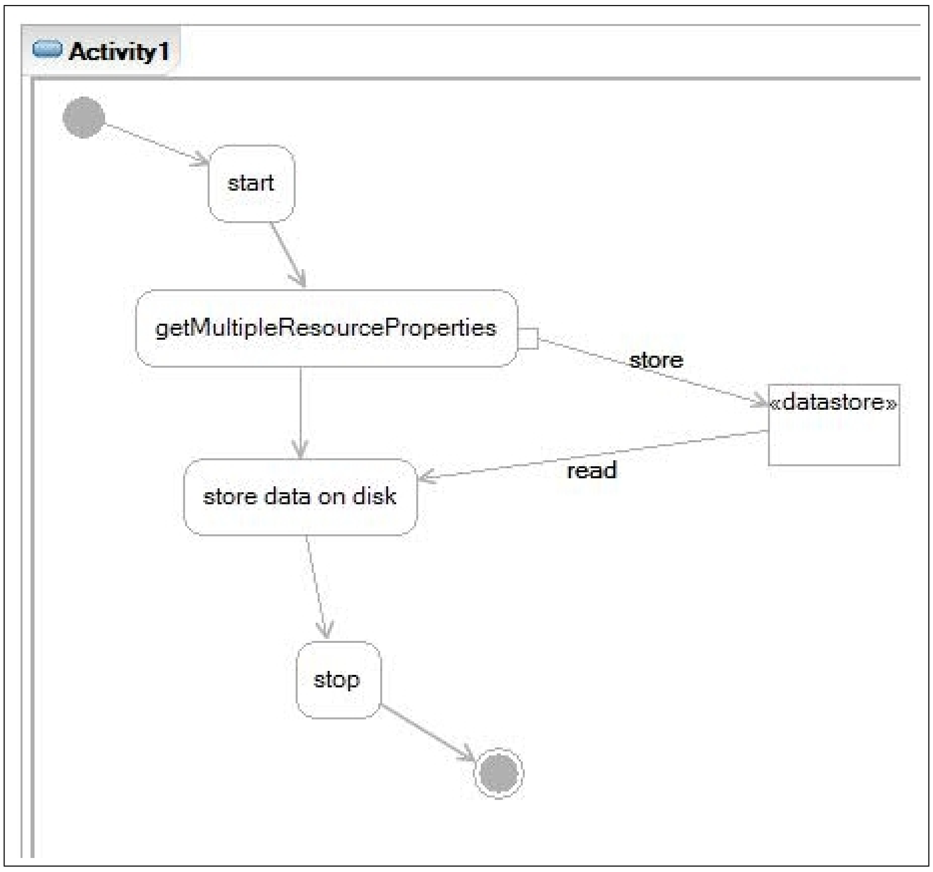


Figure 5.7: UML2 Process for accessing a WS-Resource

be achieved, as the life cycle of the partner can be modelled. In figure 5.8 the same collaboration as in figure 5.7 is modelled but it is now possible to model the life cycle of the WS-Resource, with the start message creating an instance of the diskdrive and the stop message destroying the instance. The properties that are available from a stateful resource can be modelled using attributes in the interface, and class diagrams for their internal details. An example of this can be seen in figure 5.9.

5.2 BPMN

BPMN [11] is a visual language for defining Business Processes. This makes it a suitable candidate for modelling service compositions. It does only support Web Service as the service invocation technology, and therefore only the RFP case has been implemented using BPMN. The complete set of models for this

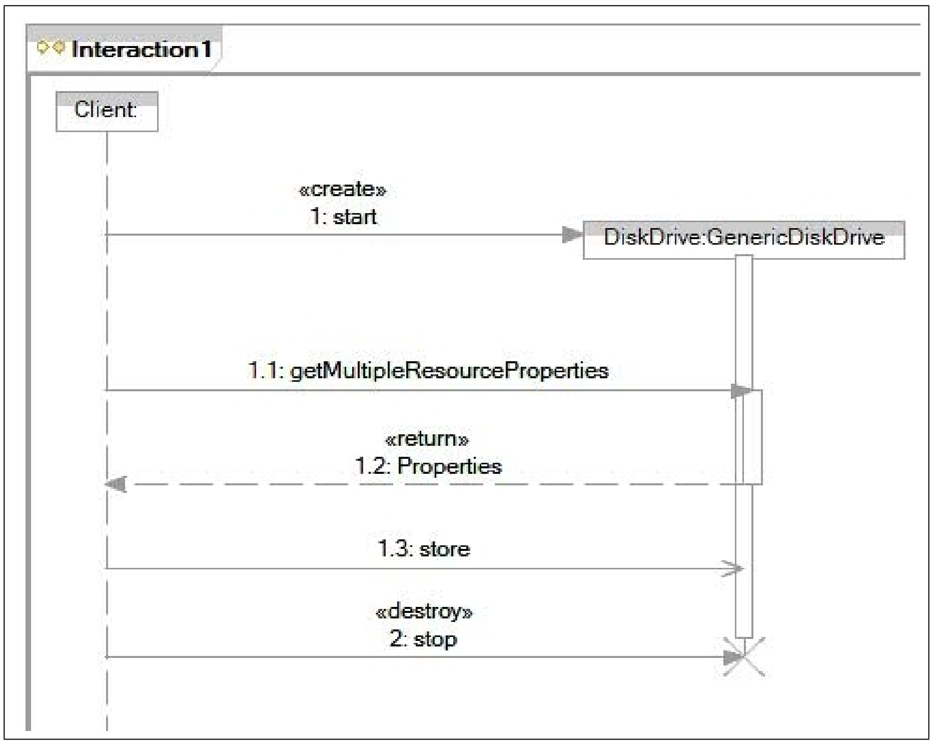


Figure 5.8: UML2 Protocol for accessing a WS-Resource. The client creates and destroys the resource, and the same instance of the resource is accessed at all times.

implementation can be found in appendix C.3. All models are created using System Architect by Popkin.

5.2.1 External Requirements

External requirements is those that concern externally visible aspects of the service composition. In BPMNa business process is modelled inside a pool, thus externally behaviour can be modelled to be between pools.

Service Description BPMN does not have support for external service description. It is possible to model that a message is received by a process, and from this a service description can be derived, but the modeller does not have explicit control of the interface of a service. The external view of a BPMN process does not specify the time dependencies, which means that it is not possible to completely specify a protocol, it is only possible to specify the message interchanges, but not the time dependencies between them. A message interchange between two pools, or partners is shown in figure 5.10, where the "winner details" message is sent to an external pool.

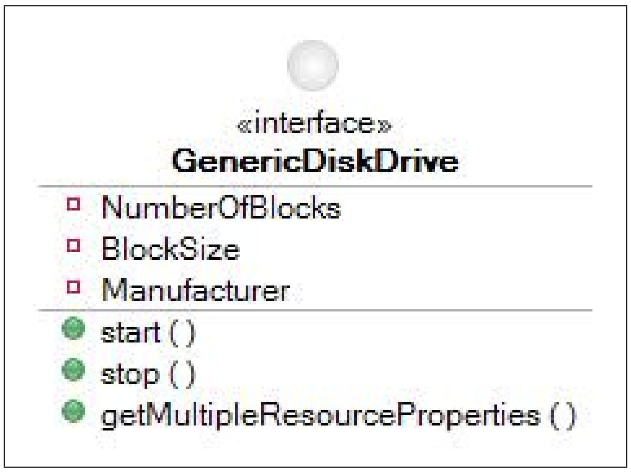


Figure 5.9: UML2 Interface with both operations and attributes. This interface contains public attributes such as BlockSize and operations such as start().

Partner Definition Partners can be defined in BPMN, by specifying them as business processes in external pools. These processes can be specified from an external viewpoint, focusing entirely on the externally visible behaviour. As there is more difficult to specify a service interface explicitly the partner requirements must be derived from the message interchanges between the partners in the process definition.

Instance Access Management Instance management is an aspect that is not supported by the BPMN specification.

5.2.2 Internal Requirements

Inside a pool in BPMN, a graph of tasks is used to model the behaviour of the business process.

Expressiveness - 5 basic workpatterns BPMN supports most of the identified workpatterns as is shown in appendix B, and the case implementations shows that the 5 basic workpatterns (sequence, parallel split, synchronization, exclusive choice and simple merge) are supported. [61] models all workpatterns found by [58] using BPMN.

Data Manipulation Dataflow inside a process is not well supported in BPMN. This must be modelled using the data object artifact, which is associated with one or more tasks, instead of flowing between tasks. This approach is document based rather than based on traditional data objects. A manipulation of data object artifact could be performed inside as a specially defined task, although there are no specific support in the BPMN for specifying such a data manipulation

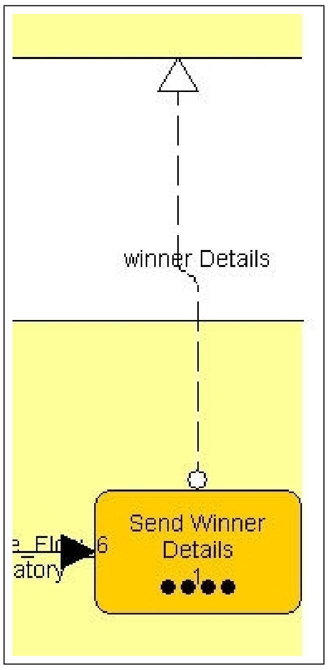


Figure 5.10: BPMN Message Flow. The "winner Details" message is sent to an external partner.

expressions. Figure 5.11 shows an example from the BPMN RFP-case implementation where the "supplier list"dataobject is used by the "send rfp"-task.

Communication A communication task in BPMN is defined to be a Web Service invocation. These kinds of tasks are restricted to the communication paradigms that are supported by the web service standards, and only support the consumptions of stateless Web Services. No task types are defined for receiving a message. Communication is also modeled in BPMN using events, specifically the message event, seen in figure 5.12. This can be used to model both sending or receiving a message, and as these events can be occur both as starting, intermediate or final events, they are good candidates for modelling message interchanges. In figure 5.12 an message event models the sending of an message to another partner. This event is externally visible, but there can be more detailed behaviour leading to the sending of this message which is hidden in the external viewpoint.

Conversational Services Business processes are conversational in its nature, which makes, a specialized business process language such as BPMN suitable for modelling a conversation, at least at a high abstraction level. At a more detailed level, some technical requirements are not as well supported. This includes the ability to model dynamic endpoint management and instance management. In the case model it is not possible to represent that the endpoint information,

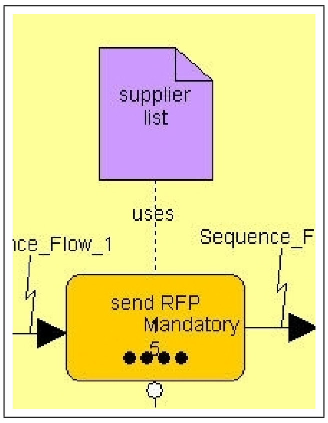


Figure 5.11: BPMN Data Object. The "send RFP" tasks uses the Supplier List data object.

such as addresses for the client, are sent in the first message to the RFP service and used for sending messages back, using the an endpoint pointing to a service hosted by the client.

Dynamic Service Selection BPMN does not support a dynamic service selection. A web service invocation tasks needs the endpoint address and other invocation information explicitly set at design time. If one wants to use a service registry this must be modelled as a normal service invocation and added to the service composition.

5.2.3 Other Requirements

Modularity A task in BPMN can be decomposed into a new process containing a set of tasks. The inner working of this task can be connected with its environment by using events. These events can be initial, intermediate or final, and can thus be used to model a that some of the expected external behaviour of a process, without revealing the inner details.

Consistency BPMN uses only a single model view, which in some cases helps maintain consistency, as all parts of the model are available at all times, but on the other hand there is a problem with the lack of separate views for defining model artifacts, which means that these must be defined in every task using the artifact. This overlap of definition makes it hard to maintain consistency, as the same artifact is defined several times..

5.2.4 Heterogeneous Services Requirements

BPMN does only support Web Services as service invocation technology. That means that as long as a service is provided as a Web Service the service invocation

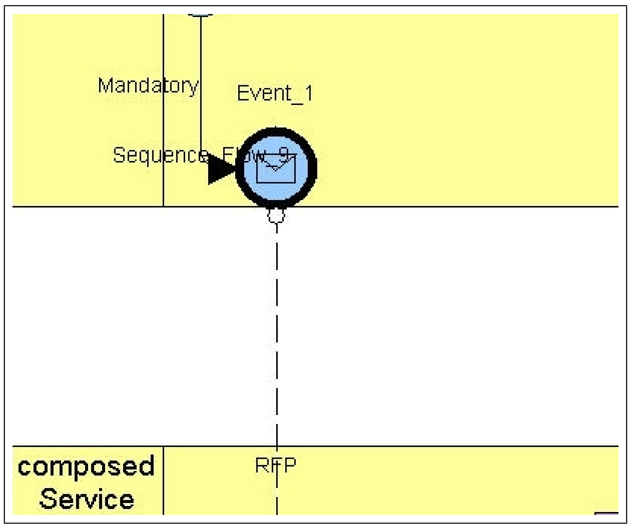


Figure 5.12: BPMN Message Event, used to send a message to an external partner.

can be modelled using BPMN.
The fact that new task types can be added means that tasks can be defined to support any service invocation technology. There is however no way of visually modelling different service invocation paradigms. If one accesses a resource on a grid that needs life cycle management, the stateful resource will be modelled in the same way as a stateless service. Resources containing properties is not possible to model in BPMN.
P2P network that requires a search for peers before invocation is better supported as an task in BPMN can contain several message exchanges. P2P technologies can use other service invocation technologies than Web Services which make it impossible to use BPMN to model composition of this type of services.

5.3 Summary of UML2 and BPMN Evaluation

UML2 - Service Composition All the model views available in UML2 makes it a very open language, with the ability to model almost any kind of system. It does however lack the structure needed to create models for a specific kind of system such as service composition. This makes it difficult to create *consistent* and *modular* models. The multiple model views makes UML2 very good at visually representing all aspects of the system, both structural and behavioural, but there are some possible inconsistencies between the different model views, specifically in modelling the *communication* with external partners. This is even more evident when the services are *conversational*, thus requiring *instance access management*. Lastly there is no explicit support for *dynamic service selection* in UML2, this must be modelled as a part of the logic of the service composition.
The strengths of UML2 is in the structural parts of the model, for *describing*

services and *partner requirements*, and for defining internal behaviour, the *expressiveness* of the control flow and the capabilities to handle *data*.
UML2 supports composition of heterogeneous services good, by being independent of any service invocation technologies such as *Web Services* or *P2P* based JXTA, and making it possible to model aspects of other service paradigms such as stateful resources on *grids*.

BPMN - Service Composition As BPMN uses only one model view it lacks the possibility to model several of the aspects that can be modelled using UML2, such as *partner requirements* and *service descriptions*. Internal behaviour and *expressiveness* is well supported when using BPMN, the only exception being the support for modelling *dataflow* between tasks . There is no support for *dynamic service selection*. BPMN is particularly good at modelling *communication* with other partners, also in *conversations* as the modelling views has a good separation of partners. The fact that BPMN only supports *Web Service* as the service invocation technology, combined with reduced support for specialized *grid* requirements makes it less suitable for heterogeneous service composition.

Evaluation Table Table 5.2 summarizes the findings in this evaluation.

	Requirements	**UML2**	**BPMN**
External	Service Description	+	-
	Partner Definition	+	-
	Instance Access Management	/	-
Internal	Expressiveness (Basic Workpatterns)	+	+
	Data Manipulation	+	-
	Communication	/	+
	Conversational Services	/	+
	Dynamic Service Selection	-	-
Other	Modularity	/	+
	Consistency	/	/
WS	Web Services	+	+
Grid	Grid Resources	+	/
	Grid Life cycle Management	+	-
	Grid Resource Properties	+	-
P2P	P2P Services	+	/

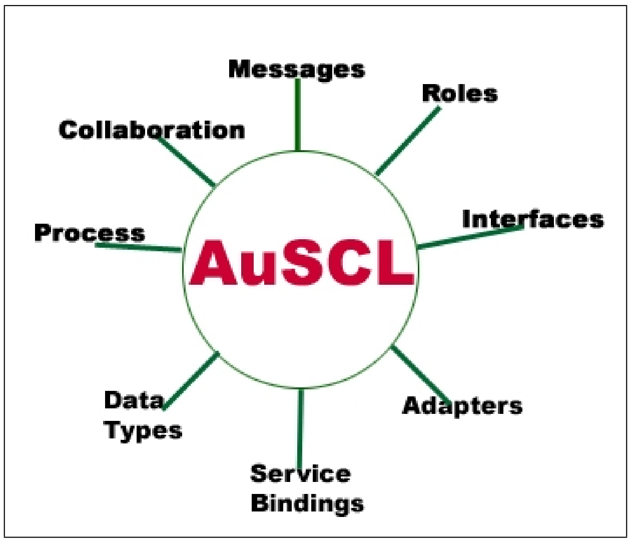
Table 5.2: Summary of evaluation of UML2 and BPMN. + is full support for the requirements, / means some support while - means no support.

Main Result UML2 lacks a structured way of presenting the separate model views, and there are some problems in modelling internal and external behaviour

using activity and sequence diagrams consistently. Heterogeneous services is supported as UML2 is platform independent.
BPMN is well suited for modelling business processes, and has good support for Web Service communication. The lack of external model views, makes it hard to model a service composition. There is problems with heterogeneous service technologies as BPMN only supports Web Services.

Chapter 6

Another unified Service Composition Language



As discussed in the previous chapter, neither UML2 or BPMN fully meets the requirements for visual composition of heterogeneous service presented in chapter 3. This chapter presents an extension to UML2 called "Another Unified Service Composition Language" (AuSCL), as a UML2 profile, which is domain specific for modelling visual service compositions.
The concepts of the profile is described in this chapter with examples from case implementations of the cases presented in chapter 4. The full case implementation models can be found in appendices C.4, D.3, E.3.
The AuSCL UML2 profile uses several of the behavioural and structural model views available in UML2. Even though each model view focuses on one specific aspect of the service composition, the AuSCL UML2 profile is designed in such a manner that by combining the model views, a complete visual model of a service composition is defined.
The model views have been divided into two groups; abstract and concrete model

views. To enhance the profile's capabilities of late binding, as much as possible is modelled in abstract models, without binding to concrete services. These service bindings are modelled separately in the servicebindings model view. To make it possible to distribute models without giving away private internal information, a separation between internal and external model views has been introduces.

6.1 Motivation for AuSCL

AuSCL is implemented as a UML2 profile. The profile enhances UML2 by adding a structure to the set of model views and defining a set of stereotypes. Since UML2 is a very open language, AuSCL narrows the possibilities the modeller have, making it easier to create a domain specific structured model of the service composition.
Modelling languages such as UML4EDOC [13] and the BPDM notation [12] uses a set of different model views to model a service composition, while BPMN [11] and JOpera [41] uses only one or two model views to represent the complete composition model. By using several model views, different aspects of the model is separated from each other, making the model more easy-to-follow, maintainable and modular.
AuSCL introduces a clear separation of abstract and concrete aspects of the model, trying to keep as much as possible abstract, to delay binding to concrete service implementations to a late stage of the development process.
The BPDM notation has a clear separation of external and internal model views. AuSCL has adopted this so that internal private models can be separated from external public models which can be published to partners in the service composition.
The service oriented principle of coarse grained services have also been taken into consideration in the AuSCL design. A service-interaction between a service provider and a service consumer can consist of several message exchanges. To model the internal process of the service composition at a higher abstraction level, AuSCL uses the service-interaction as a task in the orchestration, and details each task with a sewuence diagram, modelling the message exchanges.

6.2 AuSCL Conceptual Metamodel

The high level conceptual metamodel of the AuSCL UML2 profile can be seen in figure 6.1. This metamodel presents the relationship between the concepts used in AuSCL models, but intentionally leaves out details which are specified by the UML2 metamodel.

Abstract concepts The main construct in the metamodel is the *Service*. The *Service* is the external representation of the *Process*, and the concrete implementation of the external partners.. A *Process* consists of a graph of *ServiceInter-*

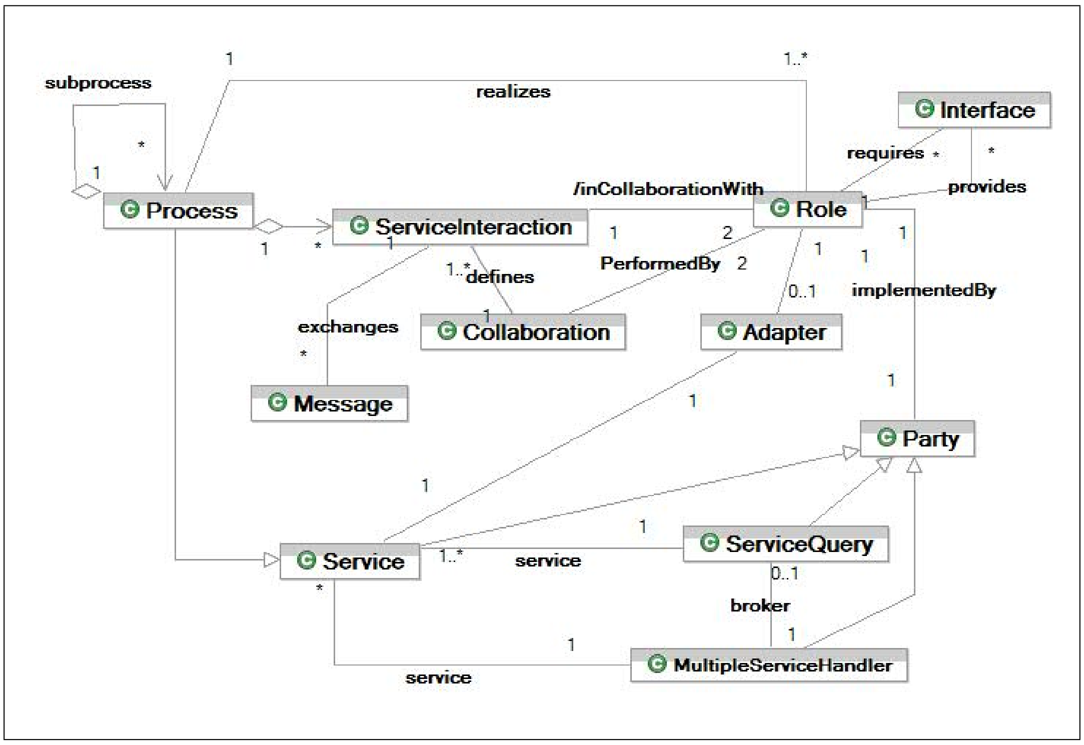


Figure 6.1: AuSCL Metamodel. This metamodel shows the concepts used in AuSCL and the relationship between them.

actions connected by control flows, and with data flowing between *ServiceInteractions*. A *Process* is defined recursively, so that a *Process* can contain several subprocesses.
A *ServiceInteraction* is defined by a *Collaboration*, as an ordered exchange of *Messages* between a pair of *Roles*. The relationship between the *ServiceInteraction* and the *Role* is derived from the *Collaboration* specifying the *ServiceInteraction*. In AuSCL a *Role* is defined to be a set of provided and required *Interfaces*.

Concrete concepts The concrete part of the metamodel focuses on handling concrete service instances. An abstract *Role* is realized by a concrete *Party*. AuSCL specifies three types of *Party*:

- *Service* - The *Party* can be a concrete instance of a *Service*, in which case the service address information is specified at design time.

- *ServiceQuery* - This *Party*-type represents the service-broker actor in the architectural model for service oriented architectures. When using a *ServiceQuery* the concrete *Service* instance will be discovered at runtime. A *ServiceQuery* specifies a service-broker instance which consists of a query and a policy for handling multiple relevant results from the query execution.

- *MultipleServiceHandler* - A *Party* capable of handling several service instances known at design time by adding a policy, which specify criteria for runtime selection of a *Service*. The *MultipleServiceHandler* can also be used in combination with the *ServiceQuery* to handle the resultset from the query.

The final concept described in the metamodel is the *Adapter*. This is used for decoupling of the abstract service descriptions given in the role-definitions and the concrete service implementations.If these do not match, an *Adapter* can be introduced to handle syntactical differences in the interfaces of the abstract definition and the concrete service implementation, such as operation or parameter names.

6.3 UML2 Profile

[62] defines an UML2-profile to be a set of limited additions to a base metamodel to adapt it to a specific platform or domain. The purpose of creating a profile is to allow limited modifications of the UML2 metamodels without requiring or permitting arbitrary changes to the metamodel. AuSCL extends the UML2 functionality by adding several stereotypes as can be seen in figure 6.2. The

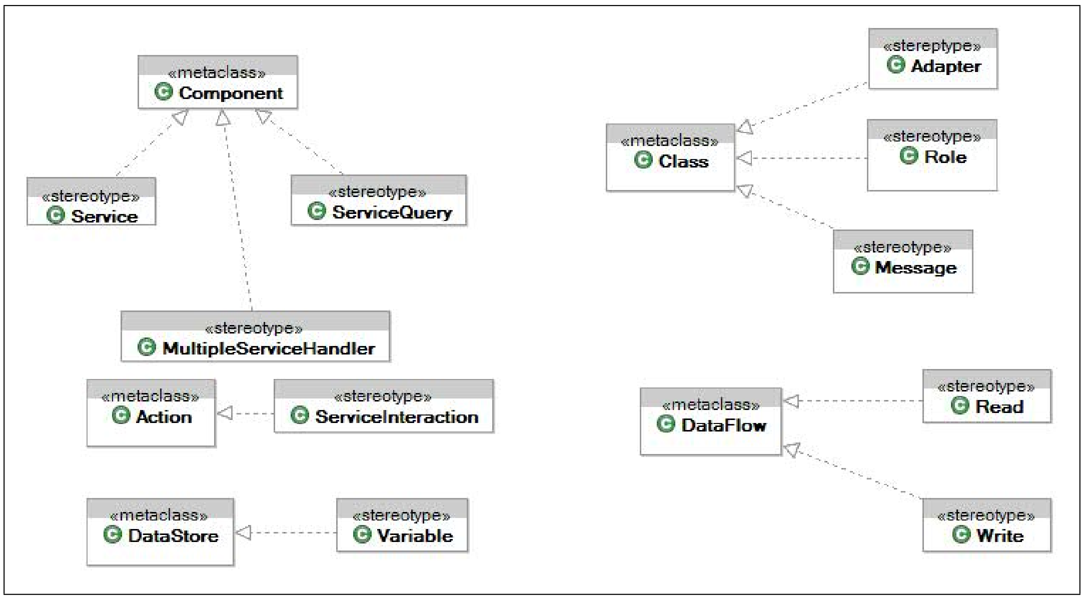


Figure 6.2: AuSCL UML2 Profile Definition. AuSCL extends the UML2 metamodel with stereotypes.

stereotypes defined in the profile are:

- «Service». A specialized *Component* representing a service. The service implements a set of interfaces. It contains an address attribute, specifying the endpoint implementing the service.

- «ServiceQuery». A *Component* representing a query to a service broker. The ServiceQuery contains an address to the broker, a query and a policy for handling multiple results.
- «MultipleServiceHandler». A *Component* for handling multiple service instances at runtime. It contains a policy for choosing which service instance to use.
- «ServiceInteraction». A specialized *Action*, which represents a interaction between to partners. The serviceinteraction is detailed by a standard UML2 sequence diagram showing the ordered set of message exchanges.
- «Variable». A *DataStore* object in an activity diagram. The datastore object is persistent throughout the scope of the surrounding activity, and can be accessed several times during one execution of an process.
- «Read». A *DataFlow* used to read from a variable. An expression can be specified to identify the sub-part of the variable to access in the read operation.
- «Write». A *DataFlow* used to write to a variable. It can contain an expression used to identify the sub-part of the variable to access in the write operation.
- «Role». A *Class* representing a role. The role specification defines the interfaces the role should provide to others, and the interfaces that the role requires from collaborators.
- «Message» A *Class*, used for messaging between roles in a collaboration. The message can contain other messages, or sub-parts defined as regular classes.
- «Adapter» A *Class* handling syntactical inconsistencies between the internal orchestration and the external partners.

UML2 Composition Viewpoints AuSCL also introduces method for structuring the model views as can be seen in figure 6.3. AuSCL relies upon several of the model views of UML2 and uses them extensively. Activity diagrams are used for describing the internal behaviour of the service composition, while sequence diagrams are used for modelling the externally visible behaviour between partners.
Class diagrams are used for modelling messages and they data being carried in the messages. Class diagrams are also used to define interfaces and roles. The binding of the abstract and the concrete parts of the models are done using component diagrams. The ports in the component diagrams represent the roles defined in the roles model view.
The model viewpoints are elaborated further in the sebsequent sections.

6.4 Model Views Structure

Figure 6.3 shows the structure of an AuSCL model. The model has two main parts; an abstract and a concrete. The abstract parts of the model has separate packages for roles, interfaces, messages, collaborations and processes.

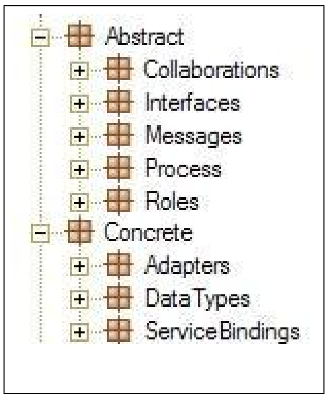


Figure 6.3: AuSCL Model Structure. AuSCL is structured in a hierarchy of packages containing model views

- Interfaces Model View (UML2 class diagram), specifies all interfaces used. Concepts from the metamodel are: Interface and Message. This model view depends on the messages model view.
- Message Model View (UML2 class diagram), specifies the messages used. Concepts from the metamodel are: Message. This model view depends on the datatypes model view.
- Roles Model View (UML2 class diagram), defines the roles in terms of provided and required interfaces. Concepts from the metamodel are: Role and Interface. This view depends on the interfaces and messages model view
- Collaboration Model View (A combination of UML2 collaboration and UML2 sequence diagrams), defines behaviour between roles. Concepts from the metamodel are: Collaboration, Role, Interface, Message and ServiceInteraction. This model view depends on the roles, interfaces and messages model views.
- Process Model View (UML2 activity diagram), defines the orchestration of the services used in the service composition. Concepts from the metamodel are: Process and ServiceInteraction. This model view depends on the messages and collaboration model views.

The concrete part of the model consists of model views for servicebindings, adapters and datatypes.

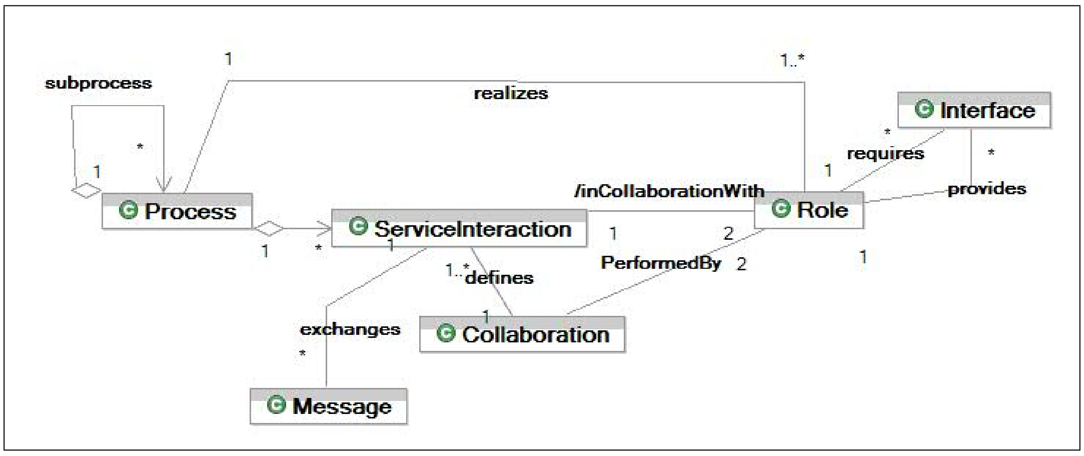


Figure 6.4: AuSCL Metamodel, abstract part. The part of the AuSCL metamodel that contains the parts used in the abstract models

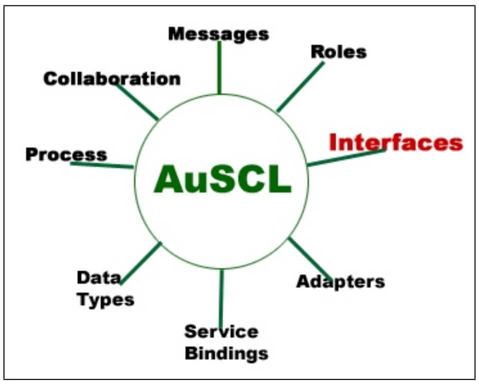
- DataTypes Model View (UML2 class diagram), defines the data carried in the messages.
- ServiceBindings Model View (UML2 Class diagram), defines the binding between concrete service instances and roles. These concepts from the metamodel is used: Role, Interface, Process, Service, ServiceQuery, MultipleServiceHandler. This model view depends on the roles and interfaces model views.
- Adapters Model View (UML2 class diagram), defines adapters to handle interface inconsistencies. These concepts from the metamodel is used: Adapter, Role and Service. This model view depends on the roles and servicebindings model view.

6.5 Abstract View

In the abstract view, concrete service instances are not used, partners in the service composition is described in terms of roles. The roles are defined with interfaces, both interfaces that a service-implementation of a role should implement, but also interfaces that is required by the service to function correctly. In this section the different model views in the abstract models will be presented. Figure 6.4 shows a subset of the metamodel, containing only the abstract parts.

6.5.1 Interfaces Model View

Interfaces are described in standard UML2 class diagrams, without any modifications. An interfaces consists of operations, with parameters and return values. An interface differs from a class in that it does not contain an implementation of



the operation only the specification of the signature of the operation. For execution of the operation described in an interface, a concrete service must implement the operation. Several operations can be grouped together in one interface and several interfaces can be grouped together to one service, which is parallel to a WSDL document for describing Web Services, where a service can consist of several PortTypes, which again can have multiple operations. The principle of service oriented architectures states that services is coarse grained and document based, and this should influence the design of these operations. If one wants to explicitly model asynchronous communication, an interface list the signals that the interface can handle, instead of listing a set of operations. Signals are asynchronous messages that are sent between partners. An interface to a stateful resource can also have attributes. The interfaces model view is dependent on the messages model view. An example of such a model can be seen in figure 6.5, where the interface both have operations and attributes.

Model Elements

- Interface - A specification of operations and signal receptions and attributes.
- Operation Signature - A specification of the parameters and return values of an operations.
- Attribute - An externally visible property.
- Parameter - A variable that is sent to the implementation of an operation when invoking it. Is defined in the messages model view.
- Return Value - A message returned to the consumer of an operation Should be defined in the messages model view.
- Signal reception - The signals that a interface is prepared to handle

UML2 Extensions This model view does not extend UML2, but uses functionality and features already defined in UML2. Signals and operations are used to support both synchronous and asynchronous communication.

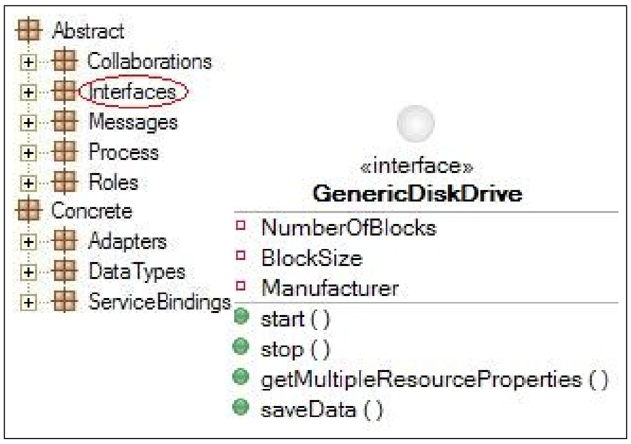
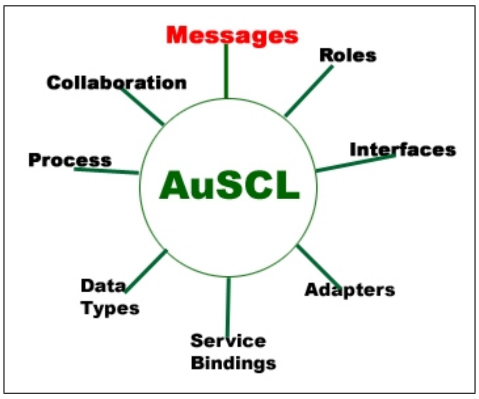


Figure 6.5: AuSCL Interface Model. An interface containing several attributes and operations, from the generic disk drive case.

Example Figure 6.5 shows an interface for an GenericDiskDrive used in the disk drive case, see appendix D. This interface describes a stateful resource, having both operations and attributes. The attributes, NumberOfBlocks, BlockSize and Manufacturer, can be retrieved using the getMultipleResourceProperties operation. The interface also has operations for life cycle management using the start and stop operations. An interface in UML2 is a stereotyped class, and the ball-symbol is the normal notation for an interface.

6.5.2 Messages Model View



Messages are defined using class diagrams, but should be stereotyped «message» to distinguish them from other classes that can be datatypes that are part of messages. A message definition is separated from the datatype definition to make it possible to use the messages when creating an abstract model, without knowing the detailed implementation of the payload of the messages. A signal is a stereotyped class, «signal», but they are also stereotyped as messages. A

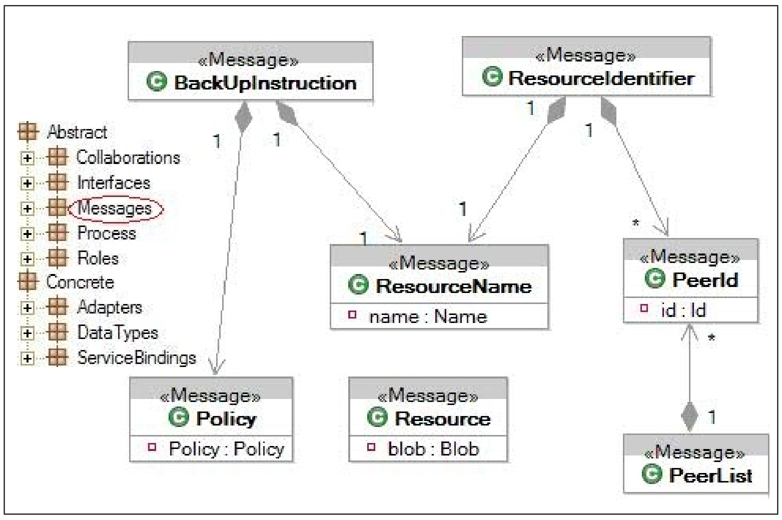


Figure 6.6: AuSCL Message Model. A class diagram defining the messages used in the service composition from the distributed office backup case.

standard class diagrams is the basis for this model view. A model example can be seen in figures 6.6 and 6.7

Model Elements

- Messages - A class that is in communication between partners.
- Message Parts - Attributes in the message that holds the payload
- Signals - A specialized form of messages, used for asynchronous communication.

UML2 Extensions This model view extends UML2 by adding the «Message» stereotype for a class, indicating that this class is a message that is sent between partners in the service composition. Besides this, standard UML2 class diagrams are used with aggregation to indicate messages that consists of other messages, and attributes for holding the pyload of the messages.

Example Figure 6.6 shows the messages model view in AuSCL. This model view is from the Distributed Office Backup case, see appendix E. The messages are stereotypes classes, using the AuSCL stereotype «Message». Aggregation is used to show messages that are parts of other messages. In figure 6.6 the PeerList message contains zero or more PeerId messages. The PeerId message contains an attribute id, which is defined in the concrete DataTypes model view. Figure 6.7 shows another AuSCL messages model view. This uses signals, indicated in the diagram by a graphical symbol, but implemented as a «Signal» stereotype in the UML2 model. A signal can be used for asynchronous communication, with interfaces being defined to support reception of a set of specific signal.

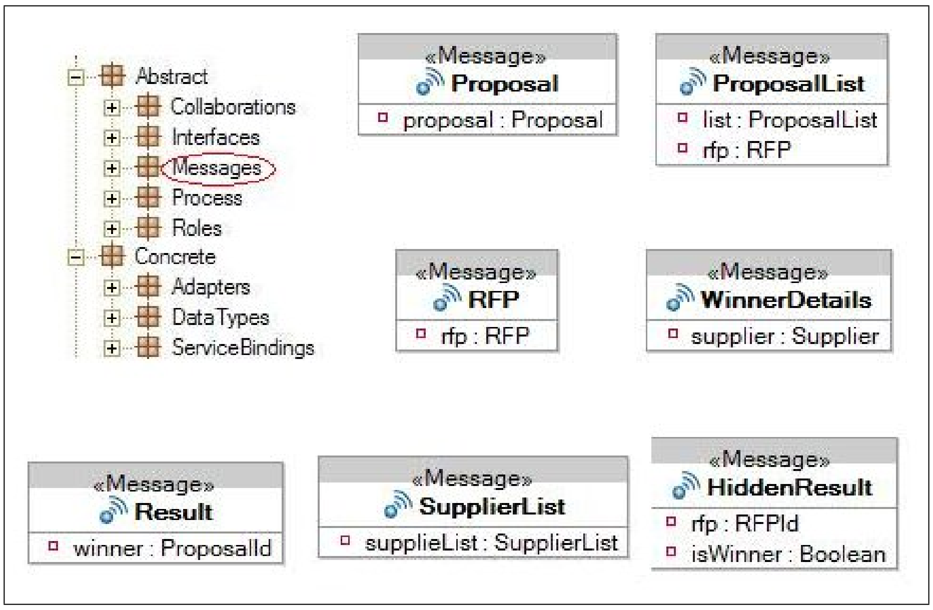
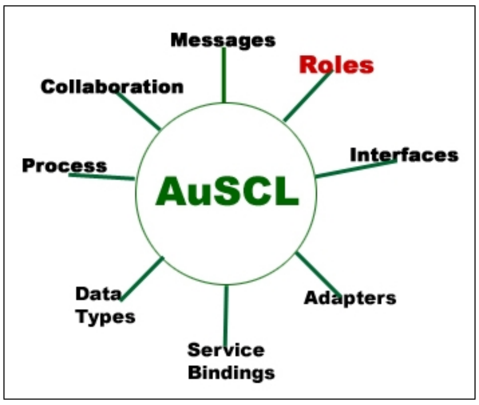


Figure 6.7: AuSCL Message Model. This AuSCL message model uses signals to model asynchronous communication

6.5.3 Roles Model View



In this AuSCL a role is a set of requirements that must be fulfilled, in terms of provided and required interfaces. A role is defined as a class with a «role» stereotype, and the model view is a standard class diagram. This model view is dependent on the interfaces model view. The roles defined in this model are used in the collaboration model view and as ports in the servicebindings model view. An example can be seen in figure 6.8, where the "generic disk drive" role implements an interface and the client role requires the interface.

Model Elements

- Role - A collection of requirements for a partner.
- Provided Interface - An interfaces that a partner provides to other partners. The interface should be defined in the interfaces model view.
- Provided Interface - An interface that a partner requires from other partners to function. The interface should be defined in the interfaces model view.

UML2 Extensions This model view uses UML2 class diagrams, but extends UML2 with the stereotype «Role», which is used to define the requirements of a partner in a serviceinteraction.The role is defined in terms of the interfaces it should provide to others and interfaces it needs others to provide. Standard stereotyped associations, «implements» and «use» are used to relate the role to the interfaces. The interfaces defined as provided and required in this model view, can also be seen in the service binding model view, where the ports use the ball/socket notation for this.

Example Figure 6.8 shows the definition of the two roles participating in the Generic Disk Drive case, see appendix D. There is only one interface used in this case, so the role definitions are relatively simple. The Client role uses the

GenericDiskDrive interface, thus being a required interface for this role, while the Generic Disk Drive Role provides the GenericDiskDrive interface.

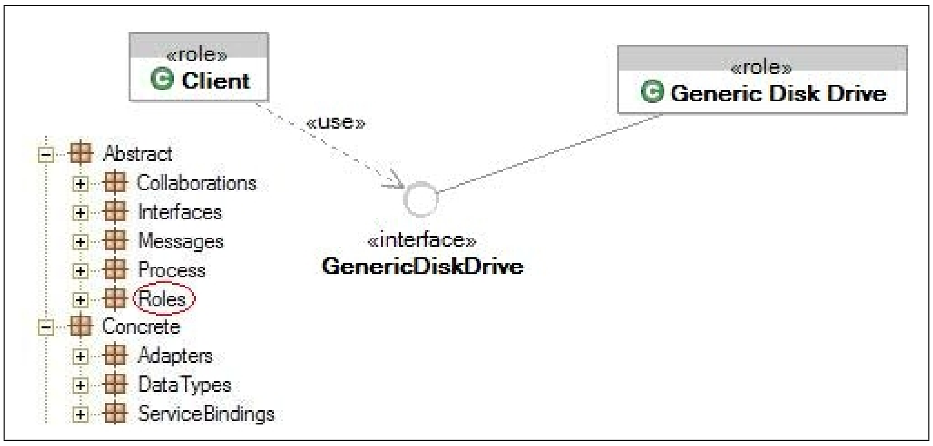
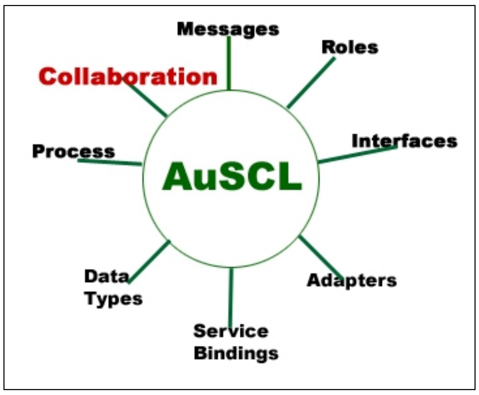


Figure 6.8: AuSCL Role Model. Provided and required interfaces define the requirements for a role.

6.5.4 Collaboration Model View



This model view models interaction between the roles to accomplish a functional objective, such as processing an order or validating a credit card. Two UML2 model views are used to model this. Collaboration diagrams are the top level which states the roles that are a part of this collaboration. A collaboration is a set of interactions between the partners, defining the sequence of messages being sent to achieve a functional objective. These message exchanges, or interactions, are implemented using the defined messages and interfaces. Such a interaction between roles is modeled using the sequence diagrams model view in UML2. An interaction between partners that achieve a specific functional objective could consist of several message interchanges. As the high level interactions or service consumptions are detailed in separate diagrams, it is possible to refer to these service consumptions before the internal details have been specified. Specifically they can be used in the process model view, to create an abstract process, and details of an service interaction can be changed without affecting the rest of service composition, decreasing the coupling between the partners. Figure 6.9 shows an high level view of an collaboration, with the roles specified. The detailed interaction between the roles are shown in sequence diagrams as in figure 6.10 where the ServiceToSupplierRegsitry role invokes and operation on the SupplierRegistry role.

Model Elements

- Role - A partner in a collaboration. The role defines the external behaviour and structure of a partner
- Message - A message is sent between partners. A message can be either synchronous or asynchronous.
- Collaboration - A definition of behaviour between partners.
- ServiceInteraction - A service consumption. An interaction is built of message exchanges. One serviceinteraction should perform a functional objective.
- Sequence Diagram - A detailed definition of the message exchanges between partners to perform a service consumption.

UML2 Collaboration and Sequence Diagrams

UML2 Collaboration Diagrams A collaboration is a description of a collection of objects that interact to implement some behaviour within a context [62]. A collaboration contains roles, and these roles represents a description of objects that can participate in an execution of an collaboration [62]. An object may participate in more than one collaboration, and also have several roles in the same participation.

UML2 Sequence Diagrams An interaction is a set of messages within a collaboration that are exchanges by roles across connectors [62]. A message is a one way communication between two objects, a flow of control with information from the sender to the receiver. A message can be a signal (an explicit, named, asynchronous interobject communication), or a call (the synchronous or asynchronous invocation of an operation with a mechanism for later returning to the sender of a synchronous call) [62]. A sequence diagram two dimensions to model this. Time moves in the vertical direction in the diagram, while the horizontal direction shows the roles. There is one column containing a lifeline for each roles participating in a interaction. UML2 sequence diagrams supports complex control flow by using the combined fragments features. These can be a link to another sequence diagram, or control-flow constructs such as loops,b conditional or parallel flows.

UML2 Extensions There are no extensions to UML2 used in these models. The high level collaborations diagrams shows the name of the collaboration and the roles participating. The participating roles are defined in the role model. The detailed interaction diagrams uses standard UML2 notation to show how the roles interact with message-exchanges to perform some task. The lifelines are typed according to the roles that are participants in the collaborations.

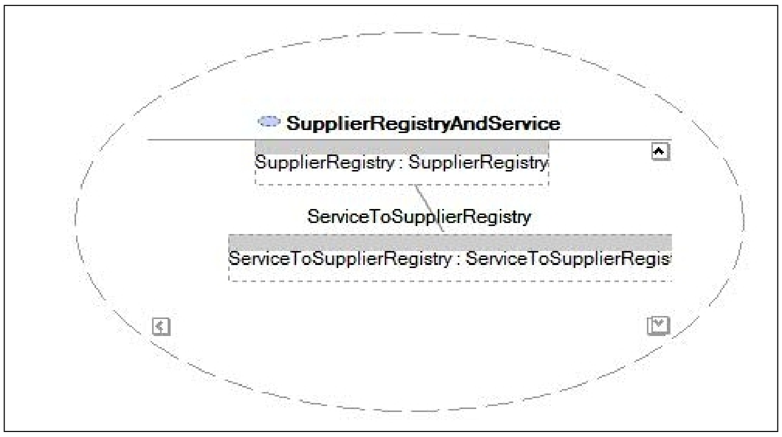

SupplierRegistryAndService
SupplierRegistry : SupplierRegistry
ServiceToSupplierRegistry
ServiceToSupplierRegistry : ServiceToSupplierRegis

Figure 6.9: AuSCL Collaboration. A high level definition of a Collaboration, showing the participants.

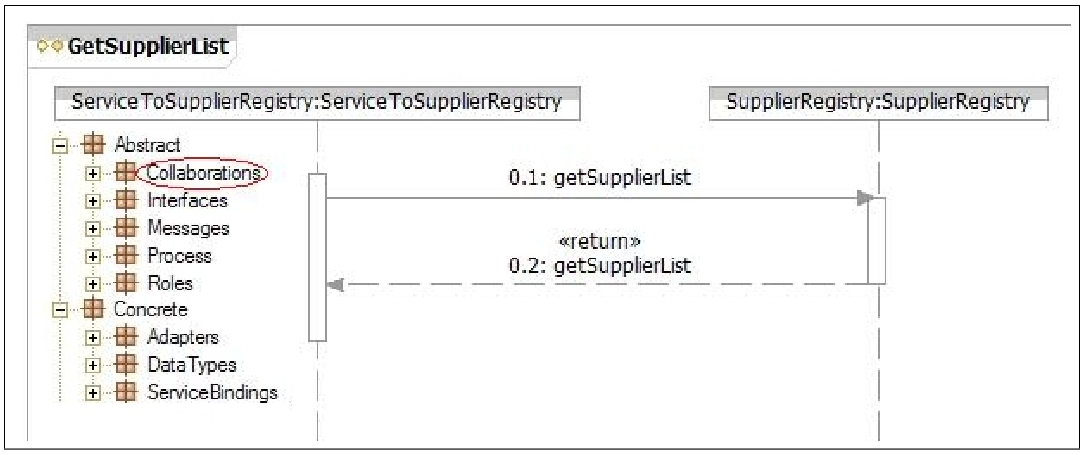
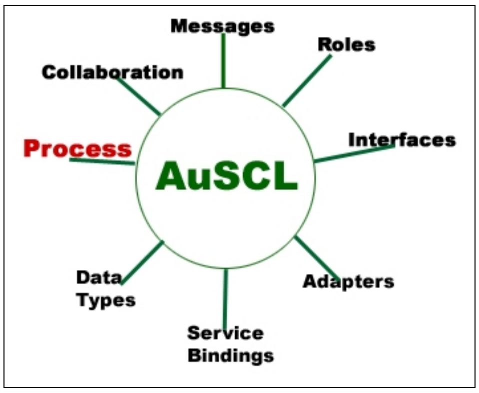


Figure 6.10: AuSCL Service Interaction. An AuSCL Serviceinteraction shown as a sequence diagram

Example Figures 6.9 and 6.10 shows the two model views used in defining the collaborations for the RFP case, see appendix C. Figure 6.9 is the high level diagram, showing the roles participating in a collaboration, in this case the SupplierRegistry and the ServiceToSupplierRegistry. The collaboration is also given a name. Figure 6.10 shows the behaviour for one serviceinteraction in the collaboration. The roles are represented as lifelines in the sequence diagram. Figure 6.10 shows a synchronous service invocation, calling the operation get-SupplierList, getting a supplierList message in return. The operation signature is defined in the interfaces model view, and the messages being passed are defined in the messages model view.

6.5.5 Process Model View



This is the internal definition of the execution of the service composition, described in a UML2 activity diagram. It defines the flow of service interactions, and the data dependencies between these in an activity graph. A service interaction is a stereotyped action, and is detailed by the interaction from the relevant collaboration. The data in process is stored in variables, which are stereotyped datastore objects. Such a variable are visible in the scope of a process, and is destroyed when the process is finished.
The control flow can include other UML2 constructs such as fork/join for parallel behaviour and decision/merge for conditional flow. The dataflow can be stereotypes to «read» or «write», where a read reads the value of a variable, and a write writes the value of a variable. These can have expressions in them to constrain these operations, such as reading only a part of a variable or writing to a part of a variable. This constraints can be expressed in OCL. The data definition in the process should be unified, and any incompatible datatypes should be transformed by the adapter. Thus there are no need for datatransformations in the process directly.
A process model can be seen in figure 6.11, where the tasks, labeled «ServiceInteraction» are interactions with the other partners.

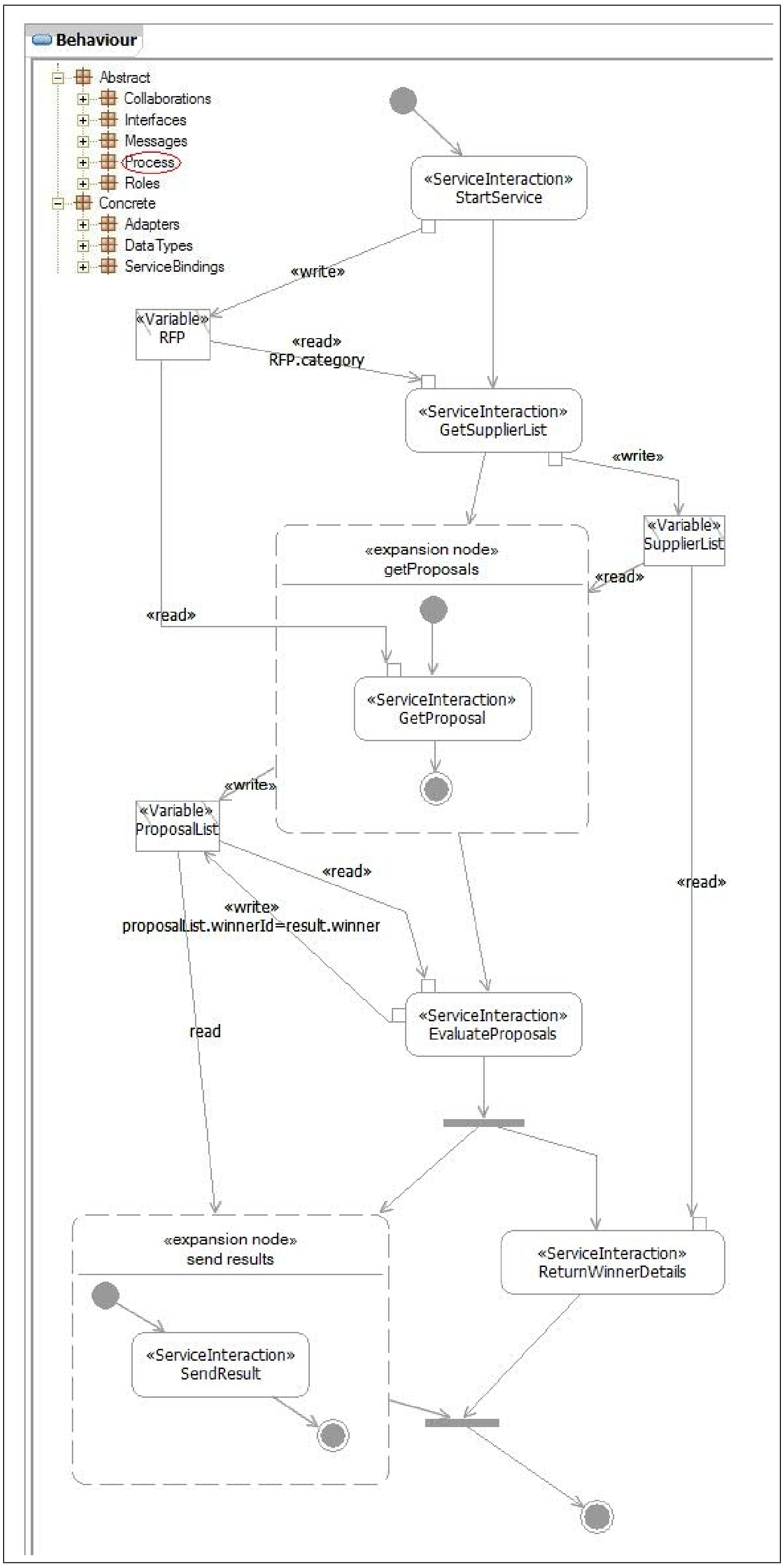
Model Elements

- Service-interaction - A service consumption defined in the collaboration. Used in the process as an action, representing the interaction defined in the collaboration model view.
- Variable - Data in a process. Can be used by several service interactions.
- Data Flow - Can be either «read» or «write», and models accessing the variables of a process.
- Control Flow - This defines the dependencies between the service interactions

UML2 Activity Diagrams An activity is a graph of nodes and flows that shows the flow of control (and optionally data) through the steps of a computation [62]. The nodes in an activity represents the a step in a workflow. Flows in the activity activates when an node is completed, and a node cannot execute before all incoming flows are activated. Nodes can be nested, to create an hierarchical structure. Special nodes are defined to create conditional flow as well as parallel flow. Data can flow from node to node, as volatile objects, or be defined as datastores, which stay active inside the scope of the activity. While a simple dataobject only flows from the output of a node to the input to another node, the datastore can be accesses several times by different threads of control. Ultimately the leaves of an activity are actions [62]. An action is a basic predefined activity, such as sending or receiving messages.

UML2 Extensions This model view uses UML2 activity diagrams and extends it by adding the «serviceinteraction» stereotype to create a specialized form of action. This action is an abstraction of the of the interaction modelled in the detailed collaboration diagram. Other extensions to UML2 are the «variable» datastore object and the «read» and «write»dataflow stereotypes. The «variable» is a data object that is persistent in the scope of the process, while the «read» and «write» variables indicate data interactions between the variable and the activity. The dataflows can also use constraints to detail what parts of the dataobject that should be accessed. Other native UML2 concepts such as pins and expansion nodes are used. Pins indicate dataflow coming out of or going into an activity while expansion nodes are a construct for modelling looping over a collection of objects. When a collection of objects is sent to an expansion node, the behaviour defined inside is performed for each element in the collection.

Example Figure 6.11 models the internal behaviour of the service composition in the RFP case, see appendix C. The control flow of the process in this example is quite simple, with mostly sequential behaviour. The process uses several variables, which are accesses throughout the process, using «read» and «write» dataflows. Examples can also be seen of specialized dataflows only using parts of the variables. The serviceinteraction GetSupplierList reads the category part of the RFP variable, and uses this part as its input. The «serviceinteraction» activities are connected to the interactions defined in the collaboration model view. The GetSupplierList activity is detailed in the model view shown in figure 6.10. The activity is thus used to model an interaction with a partner that includes several message exchanges. The expansion node construct in UML2 is used to model a loop, performing an activity for each element of a collection.



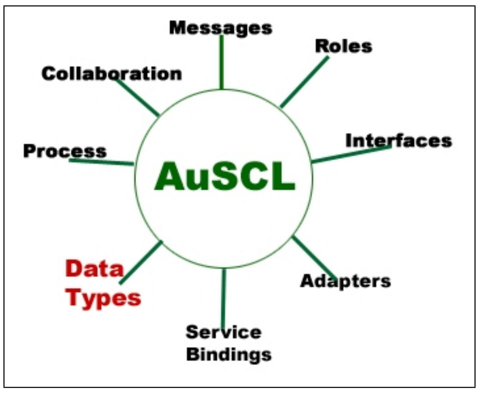
Behaviour
Abstract
Collaborations
Interfaces
Messages
Process
Roles
Concrete
Adapters
DataTypes
ServiceBindings
«ServiceInteraction» StartService
«write»
«Variable» RFP
«read» RFP.category
«ServiceInteraction» GetSupplierList
«write»
«Variable» SupplierList
«read»
«expansion node» getProposals
«read»
«ServiceInteraction» GetProposal
«write»
«Variable» ProposalList
«read»
«write» proposalList.winnerId=result.winner
«read»
read
«ServiceInteraction» EvaluateProposals
«expansion node» send results
«ServiceInteraction» SendResult
«ServiceInteraction» ReturnWinnerDetails

Figure 6.11: AuSCL Process. An AuSCL process showing the internal behaviour of a service composition.

6.6 Concrete View

The concrete view binds the abstract view to the service instances that are used in the service composition. The concrete view builds on the abstract view and must be consistent with its details.

6.6.1 Datatypes Model View



This model view is a basic class diagrams, defining the internal structure of the messages defined in the messages model view. The separation of messages and datatypes decreases the coupling as changes in the datatypes can occur without affecting the other parts of the model. Figure 6.12 shows a model of some datatypes that are used by the messages in the abstract model.

Model Elements

- Classes - Datatypes.
- Aggregation - A connection between datatypes that define that the datatype is a part in another datatype.

UML2 Extensions This model view is a standard UML2 class diagram. The classes in this model view models the internal details of the messages defined in the messages model view. No extensions to UML2 are needed in this view.

Example Figure 6.12 shows the datatypes model view from the RFP case, see appendix C, and contains all the data that is used by the message definition s in the abstract messages model view. The datatypes can have attributes and operations as in the ProposalList class.

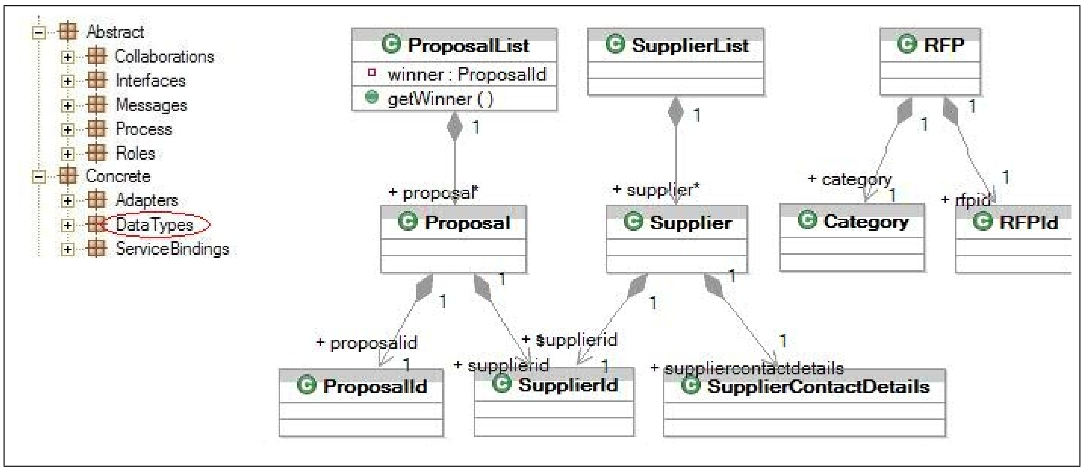
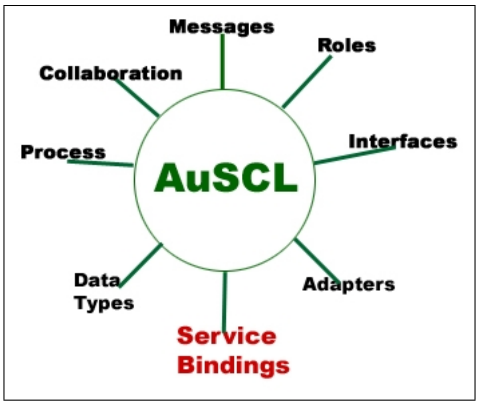


Figure 6.12: AuSCL Datatypes. A class diagram modelling the payload data of the messages defined in the message model.



6.6.2 Servicebindings Model View

This is the main model view in the concrete part of the model. Based on UML2 component diagrams, this model view creates the binding between the roles defined in the abstract view and the concrete services that are a part of the service composition. The binding is done by connecting one or more roles to a component by modelling the roles as UML2 ports, and the service instances as UML2 components. The connection between interfaces as described in the roles model view are dependencies between the provided and required interfaces on the ports. It should be noted that the partner realizing a role can be either a concrete service instance, a query to a service broker or what is called a multiple service handler. A concrete service instance component contains an address to a service description detailed enough to invoke the service. A service query contains an address to a service broker, a query for services and a policy for handling multiple results, for instance a quality of service attribute that the result set should be sorted by. A multiple service handler is a way of dealing with a situation where several concrete service instances are available, and known at design time.

Several possible methods for deciding which of these services to use is possible. One or more quality of service parameters can be used for selecting a service, or more dynamic aspects can be used, such as the first to respond to a service invocation. A multiple service handler can also be combined with a service query. Figure 6.13 shows a complete service binding model.

Model Elements

- Process - The service composition.
- Service - A component implementing a service that realizes one or more roles as required by the specification. A service needs an address attribute, containing the address of the endpoint which receivers the messages sent to the service.
- Service Query - A component acting as a placeholder for a query to a service broker.
- Multiple Service Handler - A component that hides the details of choosing between several knows service instances for a specific role realization.
- Role - Defined in the abstract model, connected to the concrete model as ports on components .
- Provided Interface - Interfaces provided by the component. Shows used the ball notation.
- Required Interface. Interfaces required by the component. Modelled using the socket notation.

UML2 Component Diagrams A UML2 component diagram consist of a set of components and interfaces that the components provide and require other components to provide. The components does not depend directly on other components, but on the interfaces that the components support [62]. The use of interfaces makes it simple to replace on component with another component have the same external characteristics, even at the instance level. Component diagrams uses the ball/socket notation to model provided and required interfaces, the balls being provided interfaces and the sockets being required interfaces. To wire two components together, having a matching pair of interfaces, a dependency is introduced from the required interface to the provided interface. Components can have ports. Ports encapsulate interaction between the contents of a class and its environment [62]. A port can also have provided and required interfaces, and messages sent to a port, which is on component is sent through to the component.

UML2 Extensions This model view uses UML2 component diagrams, extending UML to create dynamic service binding. Ports are connected to the components to model the roles that the components realize. The components are stereotyped as either «service», «service query» or «multipleservicehandler» to give the option of not binding directly to a service-instance but to the result of a query, or an element of a set of services.

Example Figure 6.13 shows the servicebinding view of the RFP case, appendix C. The components represent the concrete services, while the ports are the roles that each component realize. The connection of the service are modelled as dependencies between the provided and required interfaces. The myCustomer service has one port, showing that the customer realizes the Customer role. The customer role is defined to provide the customer interface, and require the ServiceToCustomer interface. This is shown using the ball/socket notation. These interfaces are connected with the interfaces that the RFP service component provide through its port realizing the ServiceToCustomer role.

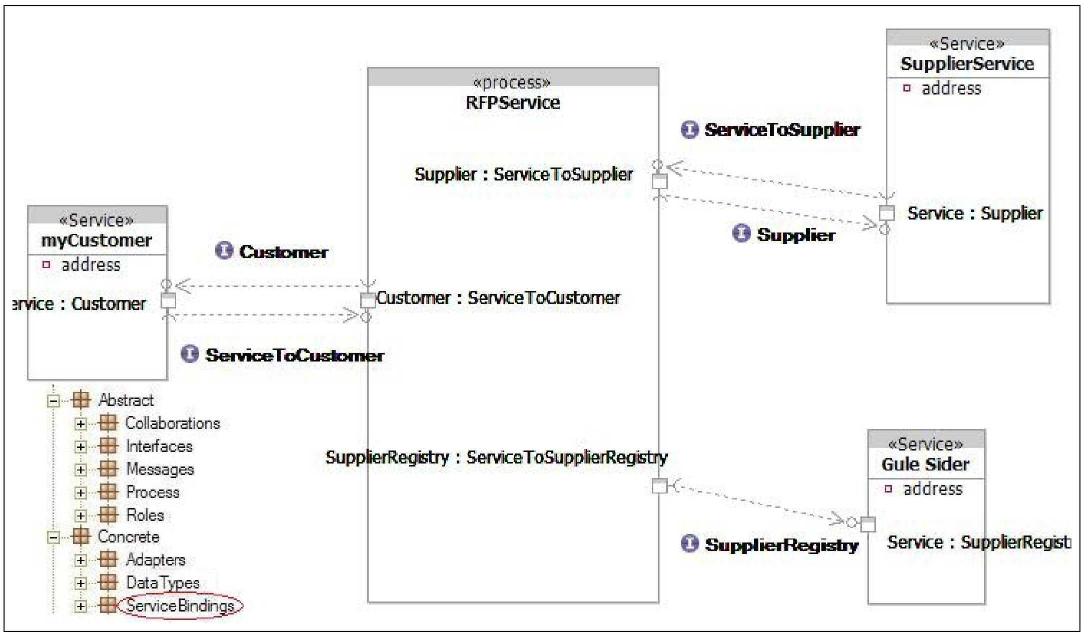
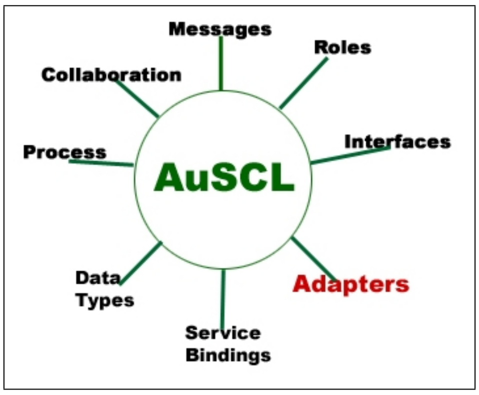


Figure 6.13: AuSCL Service Binding. The AuSCL servicebindings model view binds the abstract models to concrete service instances.

6.6.3 Adapters Model View

The interface described in the roles, and the interface of the concrete service instances realizing the roles can have differences, or they can start out as equal but change through out the life cycle of the service composition. To handle this the adapter [63] design pattern is used. This decouples the interface of the target object and the adaptee by introducing an adapter between them to



handle inconsistencies in the two interfaces. The mapping code of the adapter can be defined using OCL. Figure 6.14 shows an adapter, where the name of the operation is different in the concrete service and the defined interface in the abstract model. The mapping code is shown in the constraint of the adapter class.

Model Elements

- Adaptee - The concrete service instance.
- Adapter - The logic for mapping the targets interface to the adaptee's interface.
- Target - The role defining the interface

UML2 Extensions This model view is an instance of an design pattern. The pattern is implemented using standard UML2 constructs and does not extend UML2.

Example Figure 6.14 shows the modelling of an adapter, operating between the abstract process and the concrete service implementation to resolve any differences in interfaces. In the case shown in the example, the name of the operation is different, being defined as getSupplierList() in the abstract definition, while the relevant operation is called hentForhandlerList() in the concrete service. The defined behaviour of the adapter is then to transform the getSupplierList() invocation to a hentForhandlerList() invocation.

6.7 Main Contributions

AuSCL uses UML2 as the base for creating a visual language for service composition, building on some of the concepts from BPDM and UML4EDOC while using the features already available in UML2. This section will go through the main contributions of the AuSCL UML2 profile.

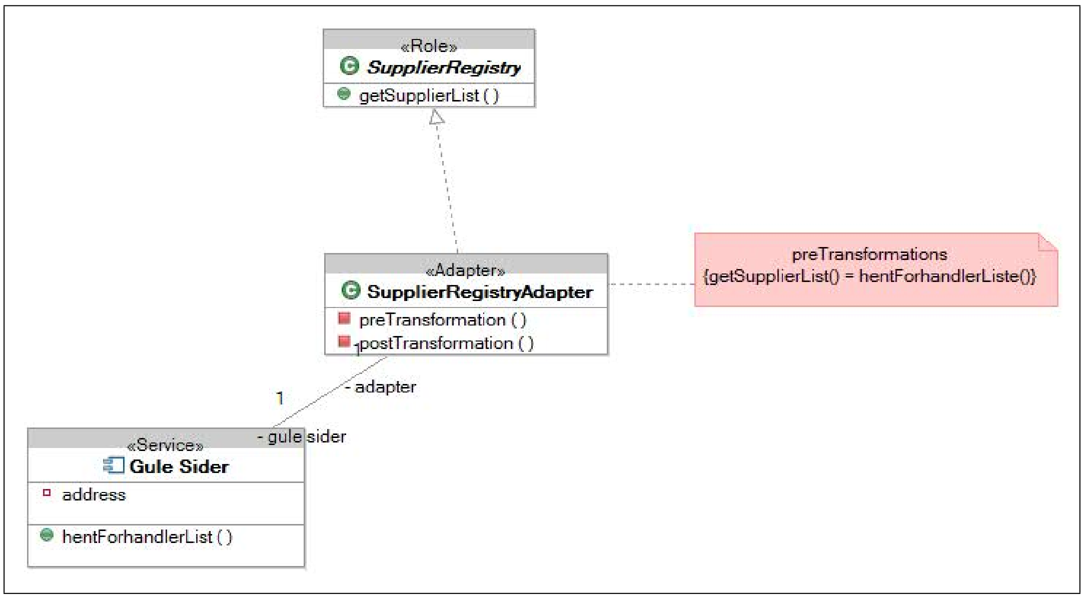


Figure 6.14: AuSCL Adapter. An adapter decouples the abstract service definitions from the concrete service implementations

Structured multiple view model AuSCL represents the model of the service composition through a set of model views, giving different viewpoints of the model. UML2 consist of a set of model views, of which AuSCL uses a subset, to give a complete representation of of the service composition. Using several different model views gives a more distinct separation of concern, with each aspect of a service composition modelled separately. Each model-construct is defined only once, and is referred to by the other model views, making it easier to create a consistent model by avoiding overlapping definitions. Multiple model views also creates a more modular model, making it possible to reuse parts of the model, for instance protocol or message definitions.

Separation of abstract and concrete models AuSCL contains an explicit separation of concrete and abstract models, all through to the structure of the model and its separate views. This separation decreases the coupling between the service composition and the details of the implementation of the services used in the service composition, making the service composition better suited to handle changes in the service implementations and making it easier to change to other services.

Separation of internal and external models By modelling external, or public, information, such as interfaces and protocols, in separate model views, these models can be used for communication with partners in the development-phase, without giving away private internal details of the system. Service descriptions can also be given as visual models, as these models are separate from the internal behaviour model views.

Adapters are used to separated the internal data model from data models used by external partners. This simplifies the the internal behaviour of the process, and increases the decoupling of the internal and external aspects of the service composition.
Internal and external behaviour is modelled consistently by using activity diagrams and sequence diagrams at different abstraction levels.

UML2 Extensions AuSCL extends UML2 in several areas, both by adding stereotypes and by introducing a structure for the created models. Table 6.1 summarizes these extensions, while table 6.2 details how AuSCL solves the weaknesses in UML2 as described in chapter 5.

UML2 Model view	**AuSCL Extension**
Class Diagrams	Added stereotyped classes to define messages and roles.
Activity Diagrams	Added stereotyped action representing a ServiceInteraction, making it possible to consistently integrate activity diagrams and sequence diagrams. Added stereotype for variable in process. By using adapters, a unified data model is used internally in the process.
Component Diagrams	Added stereotyped component for Service, Servicequery and MultipleServiceHandler.
Sequence Diagrams	Models interaction with other partners, referred to by activity diagrams.

Table 6.1: Summary of evaluation of UML2 extensions.

Structure Introducing a structure for the model views helps the modeller in the modelling process as well making it a more suitable for developing transformations in a Model Driven Architecture. UML2 is a very open modelling language, and has no defined structure for created models. The structure in AuSCL helps separating the abstract models from the concrete ones, and makes it easier to navigate in the complete model.

Stereotypes AuSCL introduces several stereotypes to extend UML2's capabilities in modelling service compositions. In UML2's structural model views, stereotypes are introduced to create specialized classes for messages and roles, as well as stereotypes for components representing services or servicequeries. In the behavioural model views, the stereotypes are introduced in the activity diagram. A specialized form of action, called serviceinteraction, represents consuming a coarse grained service, which can consist of several message exchanges. These

Requirements	**UML2**	**Solution in AuSCL**
Communication	/	Adds a method for combining Activity diagrams and sequence diagrams consistently, through the «ServiceInteraction»
Conversational Services	/	The «ServiceInteraction» construct, combing activity diagrams and sequence diagrams, makes it possible to visually represent conversation partners from an internal viewpoint.
Dynamic Service Selection	-	Adds extensions in UML2 component diagrams to define a service in terms of queries to a service broker, using the «ServiceQuery» construct, or runtime selection using the «MultipleServiceHandler» construct.
Modularity	/	Adds a structure to the model views, and a specific set of model views to use in a service composition model. Introduces a clear separation of abstract and concrete models, keeping as much as possible abstract.
Consistency	/	Adds a consistent way of combining activity diagrams and sequence diagrams for modelling communication with an external partner, by using the «ServiceInteraction».

Table 6.2: A summary of how AuSCL corrects weaknesses found in UML2.

message exchanges are modelled using sequence diagrams. Dataflow in activity diagrams are also extended, by using the variable stereotype for datastore objects, and read and write dataflows, which are constrainable so that they can access parts of an dataobject.

Chapter 7

Evaluation of AuSCL

In this chapter AuSCL will be evaluated and analyzed comparatively with the existing solutions UML2 and BPMN. There will also be a evaluation of the research challenges presented in section 1.3.

7.1 Comparative analysis of AuSCL

AuSCL is a UML2 profile and will thus have several aspects where it is similar to UML2. It is, however, aspect where the differences are more significant, and these will be highlighted here. The evaluation uses the identified requirement as a framework, presented in table 3.1.

7.1.1 External Requirements

Service Description AuSCL uses many of the same elements as UML2 when describing a service. Interfaces and classes in class diagrams are the main constructs. AuSCL uses ports in a more specific way, as roles, and a way of grouping interfaces, which give more clarity to a service description that contains several interfaces grouped together. Another aspect that is not too well supported in UML2 is support for describing the protocol in conversational services. AuSCL tries to solve this by defining collaborations between the interacting roles, which makes it possible to see for a partner the expected sequence of messages that will be exchanged.
BPMN is specifically weak in service description, as it does not have any way of describing the interface of a service in a clear manner and relies on interpretation of the behavioural model.

Partner Definition UML2 relies purely on provided and required interfaces on components when describing the requirements for a partner. AuSCL uses roles as a separate construct to describe this. The roles have both provided and required interfaces, and they are a part of the collaborations. With respect to binding a concrete service instance to a role the roles are used as ports which are connected to the components representing the concrete services. The introduction of adapters help to loosen the coupling between the partners. Again BPMN is not very strong in this area, as it does not have any model construct that support interfaces or structural modelling.

Instance Access Management There are no specific improvements from UML2 to AuSCL in this area. Instance access management is specifically important when dealing with stateful resources such as WS-resources. UML2 sequence diagrams gives the opportunity to model the state of the external partner, such as the creation and destruction of external partners can be modelled.
The other aspect of instance access management, the ability to uniquely specify which instance of a service to consume, is in some degree modelled in sequence diagrams. The sequence diagrams models the instance of the external partner, and one can model the that one two messages should be sent to the same instance of a service. This is only possible to model if messages occur in the same diagram, and is as such not a complete solution for defining this specific behaviour. Neither UML2 or AuSCL have any support for defining the exact technology used for instance management, but this is probably something that should be transparent in the visual model and handled by the execution environment. The lack of support in UML2 and AuSCL for a complete model of instance access management is down to this, as there are several possibilities for defining which instance of a service to use, but these are specific to the platform, such as WS-Addressing [42] for web services, or correlation sets for BPEL [9] based services. Possibly specific model views could have been added to AuSCL to model these specific technologies for instance access management.
BPMN does not support any modelling of state or instance control.

7.1.2 Internal Requirements

Expressiveness - 5 basic workpatterns UML2 and AuSCL are similar in this area, and the the expressiveness of UML2 as investigated in [58] also holds for AuSCL. [61] also states that BPMN has similar capabilities with respect to expressiveness as UML2. See appendix B for more details. The cases used in this thesis does not contain complex requirements with regards to expressiveness, but all 5 basic workpatterns that occur in the cases are supported by AuSCL, UML2 and BPMN.

Data Manipulation Transformation of dataobjects between tasks in the activity graph does not have a standardized method or construct in UML2. One can use a separate task to perform this, or by adding constraints to either the sending or receiving tasks, or the specific object-flow. In AuSCL this kind of data transformation is hidden from the internal process, and performed as a part of the serviceinteraction, by using adapters. The data objects, as defined in the process variables, are uniform throughout the process. This makes the internal process clearer as it focuses on just control- and data-flow, and keeps a unified

data specification in the process.
Data manipulation, in the form of getting parts of an existing dataobject does is again not supported in any standard way in UML2. In AuSCL this is done by expressions on the dataflow. These expressions define paths to parts of the variables, which are extracted and used in the serviceinteractions.

Communication One of the main weaknesses when modelling service composition in UML2 is the fact that activity diagrams are not suited for modelling message interchanges between partners. To model the sending a message to a partner, by invoking an operation, one needs to use a action to indicate that an operation has been called. To receive the response message, one needs to use another action. The partner, with which the message exchange is performed, is not visually represented in the model, but is identified with textual annotations in the model. An action can thus only represent the sending or receiving of a single message. Sequence diagrams is the best way of modelling the interchanges of messages, but there is no standard way of connecting the communication actions in the activity diagram with the message sending in the sequence diagrams.
In AuSCL the message interchanges are represented in collaborations, with the details being modelled in sequence diagrams. These interactions, which can contain several message interchanges are used as a serviceinteraction. This means that in the internal process, an action can represent a service consumption, not just an operation involving sending or receiving a single message. This makes the internal process more coarse grained, and more suited to the principle of service oriented architecture, with loosely coupled, document-based and coarse grained services.
BPMN has the capabilities of having a solution similar to the approach used in AuSCL, with tasks that can contain several message events. An event represent a service interaction, consisting of several message exchanges modelled with events.

Conversational Services Conversations require a combination of internal and external models cooperating in an consistent manner to clearly model the communication with the external partners and the internal behaviour. AuSCL does this by using sequence diagrams to details the activity diagrams, this removing the problem of sequence diagrams and activity diagrams overlapping, and modelling the same behaviour. This combination makes AuSCL well suited for modelling conversations.
One weakness in AuSCL with respect to conversational service is the ability to handle callback interfaces. If a client consuming a service at runtime specifies an endpoint address to which replies should be sent, this can not presently be defined in AuSCL. This could be solved by adding a field in the «service» component named callback, containing a boolean true/false value. If true, the address of this service is set at runtime according to the information given by the client at invocation.
BPMN combines the external and internal model views in one model to model

conversations. The internal service orchestration communicates with partners in a manner that makes BPMN well suited for modelling conversational service interactions.

Dynamic Service Selection AuSCL has a clear separation between the concrete and the abstract parts of the model. The binding between the abstract and the concrete part can be static, with concrete service instances being specified, or more dynamic by using the service query construct. The service query construct makes it possible to bind to concrete service implementations at runtime. The multiple service handler construct introduces dynamic service selection without using a service broker, but by using a set of service instances known at run time. UML2 contains the possibility of separating the concrete and the abstract models, but this is up to the modeller. AuSCL makes the modeller do this by defining what each model view should contain.
The ServiceQuery construct would be more robust if a mechanism was added to handle the situation of the query returning no results. A form of exception or other fault handling mechanism could be used to gracefully handle this, making it possible for the service composition to deal with the situation.
BPMN does not contain any support for dynamic service selection.

7.1.3 Other Requirements

Modularity AuSCL has a clear separation of what each model view should represent, and no model view should be overlapping with any other model view. This increases the modularity, as a model view that has a clear separation from the rest of the model, can be easier replaced than a model that has several complex dependencies with other model views. The structure introduced by AuSCL also aids the modularity of the model. UML2 does contain some built in support for a modular design, such as in activity diagram where an hierarchy of activities are possible and in sequence diagram. Structural model views also have good support for modularity. UML2 does, however lack the built in structure of the model views, which is supported by AuSCL.
BPMN does only have one model view, but this view can be given an hierarchical structure which makes it possible to have some degree of modularity.

Consistency AuSCL uses a set of model views, where each model view represents an aspect of the model The structure of the model views is there to avoid overlapping models, which makes it better for containing a consistent model. Each model artifact has only one definition, and this is referred to by the other model views.

7.1.4 Heterogeneous Services Requirements

Web Services The Web Service stack of standards uses a simple operation based paradigm for description and invocation. The interfaces used to described services in WSDL documents are similar to interfaces in AuSCL, and the invocation of such an operation is modelled using operation invocation in UML2 sequence diagrams. Thus, AuSCL fully supports the Web Service paradigm.

Grid Services The two main requirements presented related to grid services were life cycle management and interfaces with properties as well as operations.

Life Cycle Management Grid service needs special focus on life cycle management, and even though the resource presents a stateless service based interface to do this, it is helpful if the modelling language makes it possible to show the state of the partner. UML2 supports this partly by using sequence diagrams, where the creation and destruction of partner instances can be modelled., and also modelling that the same instance that were created by one message should be used later. AuSCL uses these abilities, making it possible to model the consumptions of stateful resources and the services they provide.
BPMN does not contain functionality for life cycle management.

Resource Properties A stateful resource on a grid will present some properties in its interface as well as operations. Both UML2 and AuSCL support this fully, as interfaces can contain both operations and properties. The type of the properties are defined in class diagrams.
BPMN does not support stateful resources with properties.

P2PServices P2P network service invocation is similar to normal service invocation. There might be more message exchanges involved as there are more emphasis on searching for the peer that perform the service. In this respect, AuSCL is helpful as it makes it easy to model a serviceinteraction to contains several message interchanges. UML2 does not support this in the same way. BPMN does support this to some degree, as it is possible for a task to contain several message exchanges.

7.2 Research Challenges

In chapter 1 a set of challenges were presented. These challenges will be evaluated and discussed here.

There are actual requirements for visual composition of heterogeneous services, that are not supported by existing solutions. Several requirements for visual service composition have been identified and were presented in chapter 3. These requirements have in this report been structured into external and internal requirements. The external requirements focuses on service description, and how the service relates to its partners, both clients invoking the service, and services that are part of the service composition. Internally the requirements are more diverse, some focuses on expressiveness, while others are

related to communicating with the partners.
With respect to heterogeneous services, in this thesis restricted to P2P network and grid based services, there seems to be very little that differentiates these service technologies from Web Services at the level of abstraction being used in visual modelling. Grids consists of resources, either stateful or not, and the WSRF [23] specification hides the complexity of dealing with stateful resources behind a standard Web Service based stateless interface, with operations for managing the life cycle of the resource. In the cases where the resources actually are stateless, they are similar to regular Web Services. Even though a stateful resource is hidden behind a set of stateless operations it can be helpful if the visual language can be used to model the state of the resource, to aid the user of the language.
P2P network services are even more similar to Web Service than grid based services, even though they are not as tied to Web Service xml-based protocols for messaging and discovery. The invocation paradigm of a P2P network service is more or less the same as in a Web Service invocation with one exception. Due to the dynamic structure of a P2P network, a search is performed to find peers providing the required service prior to invocation.
The evaluation of BPMN and UML2 in chapter 5 shows that both UML2 and BPMN fails to meet all the requirementd presetned in chapter 3.

No visual service composition language available today is suitable for modelling composition of heterogeneous services. Several visual modelling languages were investigated and are presented in appendix A. The two languages perceived to be the most suitable were selected for a more thorough study, using a case based evaluation (chapter 5). This evaluation of UML2 and BPMN showed that neither meets all the stated requirements for modelling a visual composition of heterogeneous services.

UML2 Pure UML is a very open language, with a broad range of model views and extensive expressive powers. This makes it possible to model service compositions i UML2, but there are some weaknesses that can cause problems. The internal process, modelled in activity diagrams, are not suitable for modelling communication with partners. An action is used for modelling an atomic operation such as sending or receiving an message, and such an action must be decorated with textual information to give sufficient details, including what message to send and what operation to use. The level of detail in the activity diagrams becomes high and complex, as it must model each atomic communication operation separately, even though a service consumption could involve several such operations.
Aspects such as dynamic service selection is not directly supported, and there are no distinct separation of abstract and concrete models.

BPMN BPMN is a visual notation for modelling business processes, and lacks some of the aspects that are important for service composition, such as interface description, and dynamic service selection. It is also targeted at the business user, and lacks some of the low level details which is necessary to create a complete model. The notation only supports the web service standards for service invocation, which makes it suitable for web services and grids, but might make it impossible to invoke P2P network services.

A new language called AuSCL will be suitable for modelling heterogeneous service composition. AuSCL's suitability for heterogeneous service composition has been shown in the evaluation in this chapter. AuSCL enhances UML2's capabilities in some areas such as dynamic service selection, communication and model structure and a better separation of abstract and concrete models. It also makes it easier to work with services as coarse grained functional units, by making it possible to have a service interaction in the internal process consist of several messages.
UML2 contains multiple model views, and AuSCl makes a selection of these creating a structured way of presenting these model views to give a clear modularity where each aspect of the service composition is modelled only once. Adding roles as a constructs enhances the capabilities to describe what is required of a partner in a service composition.

7.3 Summary of Evaluation

The evaluation of AuSCL is summarized in table 7.1. AuSCL introduces a structured way of modelling a service composition, using a set of model views. The structured approach helps in keeping the model *consistent* and *modular*. The multiple model views show the service composition model from several separate viewpoints, and the structure aids consistency as it avoids the creation of overlapping model views. AuSCL extends the functionality of UML2 in several areas, adding stereotypes to handle *dynamic service selection*, and combining activity and sequence diagram in a consistent manner to handle *conversations* and specify *communication* from both an internal and external viewpoint. Stereotypes are also used to extend UML2's capabilities in defining requirements for a *partner* in a service composition. No extension has been proposed to add to UML2's ability to model *instance access management*. UML2 already had good support for *heterogeneous services*, thus there have been no additions by AuSCL in this area. Other specific aspects that can be fixed is better support for callback interfaces and a more robust ServiceQuery construct.

	Requirements	**AuSCL**
External	Service Description	+
	Partner Definition	+
	Instance Access Management	/
Internal	Expressiveness (Basic Workpatterns)	+
	Data Manipulation	+
	Communication	+
	Conversational Services	+
	Dynamic Service Selection	+
Other	Modularity	+
	Consistency	+
WS	Web Services	+
Grid	Grid Resources	+
	Grid Life cycle Management	+
	Grid Resource Properties	+
P2P	P2P Services	+

Table 7.1: Summary of evaluation of AuSCL. + is full support for the requirements, / means some support while - means no support.

Chapter 8

Conclusions and Future Work

This thesis has proposed a domain specific UML2 profile called "Another unified Service Composition Language" (AuSCL). This thesis has also presented a set of requirements for visual composition of heterogeneous services, based on a set of case-implementations, investigated service composition languages and related technologies. These requirements and case studies has been the base for an evaluation of UML2[10], BPMN[11] and the proposed AuSCL UML2 profile.

8.1 Conclusions

Programming by composition can reduce the time and cost of developing software systems, by reusing already existing software systems to create new value-added functionality. A combination of Service Oriented Architecture [4] (SOA) and Model Driven Architecture [5] (MDA) can be used to define a visual language for creting models of composition of services, which can be transformed to an executable composite service.
AuSCL is such a language, implemented as a UML2 profile to enhance the functionality of UML2 and create a domain specific structure of multiple model views for a service composition model. AuSCL extends UML2 with constructs for dynamic service selection, to support late binding, and a consistent way of combining activity diagrams and sequence diagrams to model communication with external partners from both external and internal viewpoints, by using the two diagrams at different abstraction levels. AuSCL is structured to separate abstract and concrete model views, and differentiates internal and external model views.
AuSCL uses ideas from UML4EDOC [13] and the BPDM notation [12] to create a a domain specific UML2 profile for composition of heterogeneous services. A UML2 has the advanteges from UML2 of good support for metamodelling through the Meta Object Facility [37] (MOF) and a standardized lexical representation in XML Metadata Interchange [38] (XMI) helping the potential integration of AuSCL into a complete MDA platform.

Evaluation A set of requirements has been identified for a visual language for composition of heterogeneous services. These requirements are shown in table 8.1. The requirements are structured to separate requirements for general service composition from requirements related to the set of heterogeneous services considered in this thesis, Web Service, grid services and P2P-network services. The general service composition requirements are either related to external or internal aspects of the service composition as shown in the table. The presented

	Requirements	**AuSCL**	**UML2**	**BPMN**
External	Service Description	+	+	-
	Partner Definition	+	+	-
	Instance Access Management	/	/	-
Internal	Expressiveness	+	+	+
	Data Manipulation	+	+	-
	Communication	+	/	+
	Conversational Services	+	/	+
	Dynamic Service Selection	+	-	-
Other	Modularity	+	/	+
	Consistency	+	/	/
WS	Web Services	+	+	+
Grid	Grid Resources	+	+	/
	Grid Life cycle Management	+	+	-
	Grid Resource Properties	+	+	-
P2P	P2P Services	+	+	/

Table 8.1: Summary of evaluation of AuSCL compared to UML2 and BPMN. + is full support for the requirements, / means some support while - means no support.

requirements in table 8.1 are the base for the evaluation of UML2, BPMN and AuSCL, with case implementations (appendices C, D, E) to support the evaluation. The evaluation results, summarized in table 8.1 shows that aspects of BPMN and UML2 can be improved to enhance the support for visual composition of heterogeneous services.
The evaluation of **BPMN** showed that with only one model view and no structural model views, and the lack of ability to visually define message and interfaces, BPMN does not meet the requirements for a visual service composition language. In relation to heterogeneous services, BPMN is bound the Web Services as the invocation technology, and does not support modelling of the life cycle or stateful properties of a grid service. BPMN is strong when modelling communication between partners and has a consistent link between internal and external behaviour.
UML2 has a set of model views available, making it a very open language. With several model views different aspects of the model can be modelled separately, but the no built in structure exists for a specific domain such as composition

of heterogeneous services. UML2 meets the requirements for heterogeneous services, with the ability to model life cycle management and resource properties for grids, and being platform independent in relation to the invocation technology being used. UML2 does not have a well defined link between the internal and external definition of the behaviour, specifically in terms of communication between partners. This is modelled both using activity diagrams and sequence diagrams, and the combination of these to diagrams can create two specifications of the same behaviour. Other aspects of UML2 which can be improved is to introduce dynamic service selection, for late binding to concrete service implementations.
The evaluation of **AuSCL** shows that AuSCL improves upon the weaknesses

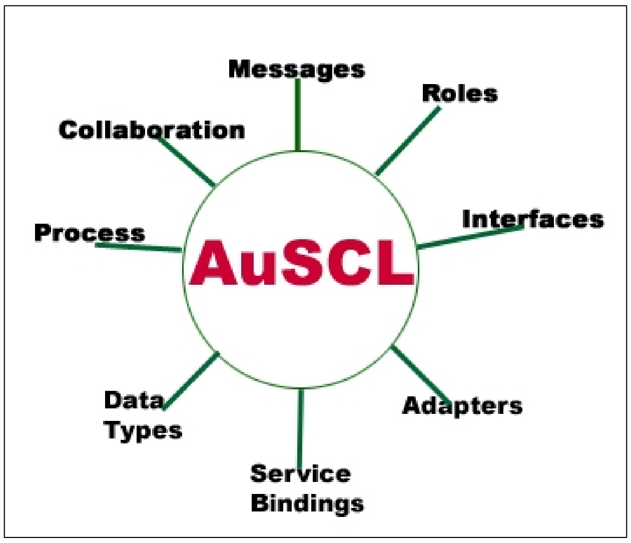


Figure 8.1: AuSCL uses several model views to model a service composition.

discovered in UML2. Dynamic Service selection is added by a set of stereotypes, making it possible to define a service by its address, by a query to a service broker or by a handler selecting between a defined set of services. Communication between partners are modelled with a combination of activity and sequence diagrams, but the serviceinteraction stereotype is introduced in the activity to have a coarse grained activity which is detailed by a sequence diagram specifying the message exchanges.
The structure introduced in AuSCL separates between abstract and concrete models, and introduces a clear separation of concern, with each model view modelling a specific aspect of the service, and each construct in the model being defined only once. Partners are described in terms of roles in the abstract model views, and bound to concrete service implementations by the concrete models. Internal and external information is separated, making it possible to share the external models with partners without giving away implementation details.
The structure helps the developer as decisions has already been made about what model views to use for modelling a service composition, thus narrowing down the options available in UML2. Modularity is also increased by using the structure

as a predefined set of model view makes it possible to reuse a specific model view in other service composition models.

8.2 Future Work

Several aspects of AuSCL can be the base for future work.

- The profile can be implemented in a tool, creating a visual representation for the stereotypes defined by AuSCL.
- Implementations to fix the identified weaknesses; callback-interface support in the servicebindings model view and service query robustness should be added.
- AuSCL can add support for semantic description of service, making it possible to create goal-driven automatic composition.
- AuSCL can support some visual profile for Quality of Service, using it for QoS based dynamic service selection.
- Transformation definitions from AuSCL to a executable service composition language such as BPEL or JOpera can be defined.
- AuSCL can be modified to support the Business Process Definition Metamodel, when a standardized version of this is presented. This would automatically add support for transformation to other languages also supporting BPDM
- Other service technologies or paradigms, such as agent-based services can be studied to see if they create additional requirements that might create a need for futher enhancements to AuSCL

Bibliography

[1] M. D. McIlroy. (1968) Mass produced software components. [Online]. Available: http://www.ericleach.com/massprod.htm

[2] G. Wiederhold, P. Wegner, and S. Ceri. (1992) Towards megaprogramming. [Online]. Available: http://www-db.stanford.edu/CHAIMS/Doc/Papers/92cacm/cacm-final.ps

[3] (2005) Sodium home page. [Online]. Available: http://www.atc.gr/sodium/project.asp

[4] Z. Stojanovic and A. Dahanayake, *Service-oriented Software System Engineering Challenges And Practices.* Independent Pub Group, 2005.

[5] OMG. (2005) Model driven architecture homepage. [Online]. Available: http://www.omg.org/mda/

[6] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen. (2003) Soap version 1.2 part 1: Messaging framework. [Online]. Available: http://www.w3.org/TR/soap12-part1/

[7] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. (2001) Web services description language (wsdl) 1.1. [Online]. Available: http://www.w3.org/TR/wsdl

[8] L. Clement, A. Hately, C. v. Riegen, and T. Rogers. (2005) Uddi version 3.0.2. [Online]. Available: http://uddi.org/pubs/uddi_v3.htm

[9] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, and J. Klein. (2003) Specification: Business process execution language for web services version 1.1. [Online]. Available: http://www.ibm.com/developerworks/library/ws-bpel/

[10] OMG. (2004) Uml2 superstructure specification. [Online]. Available: http://www.omg.org/cgi-bin/doc?ptc/2004-10-02

[11] S. A. White. (2004) Business process modeling notation (bpmn) version 1.0. [Online]. Available: http://www.bpmn.org/Documents/BPMN%20V1-0%20May%203%202004.pdf

[12] J. H. Frank, T. A. Gardner, S. K. Johnston, S. A. White, and S. Iyengar. (2004) Business processes definition metamodel concepts and overview. [Online]. Available: www.bpmn.org/Documents/BPDM/BPDM%20Whitepaper%202004-05-03.pdf

[13] B. Wood. (2001) A uml profile for enterprise distributed object computing (edoc). [Online]. Available: http://edoc.doc.ic.ac.uk/pdf/BryanWood.pdf

[14] D. Quartel, R. Dijkman, and M. v. Sinderen, "Methodological support for service-oriented design with isdl," in *2nd International Conference on Service Oriented Computing*, New York City, 2004.

[15] R. Dijkman and M. Dumas, "Service-oriented design: a multi-viewpoint approach," *International Journal of Cooperative Information Systems*, vol. 13, no. 4, pp. 337–368, 2004.

[16] C. Kaler. (2002) Web services security. [Online]. Available: http://www-106.ibm.com/developerworks/webservices/library/ws-secure/

[17] IBM. (2004) Web services transaction. [Online]. Available: http://www-128.ibm.com/developerworks/library/specification/ws-tx/

[18] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to- peer architectures and applications," in *First International Conference on Peer-to-Peer Computing.* IEEE Computer Society, 2002.

[19] M. P. Papazoglou, J. Yang, and B. J. Kramer, "Leveraging web-services and peer-to-peer networks," in *CAiSE 2003*, 2002.

[20] (2005) Jxta programmers guide. [Online]. Available: http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf

[21] (2005) Project edutella homepage. [Online]. Available: http://edutella.jxta.org/

[22] (2005) Gnutella homepage. [Online]. Available: http://www.gnutella.com/

[23] K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. V. Hewlett-Packard). (2004) The ws-resource framework. [Online]. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[24] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *Lecture Notes in Computer Science*, vol. 2150, 2001.

[25] M. Haynos. (2005) Perspectives on grid: Using automation effectively within a grid infrastructure. [Online]. Available: http://www-106.ibm.com/developerworks/grid/library/gr-automation/

[26] D. Gannon, “Programming the grid: Distributed software components, p2p and grid web services for scientific applications,” in *2nd International Workshop on Grid Computing*, Denver, 2001.

[27] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, and P. Vanderbilt. (2003) Open grid services infrastructure (ogsi). [Online]. Available: http://www-unix.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf

[28] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke. (2004) From open grid services infrastructure to wsresource framework: Refactoring and evolution. [Online]. Available: http://www.globus.org/wsrf/

[29] T. Fahringer, S. Pllana, and A. Villazon, “Agwl: Abstract grid workflow language,” *Lecture Notes in Computer Science*, vol. 3038, pp. 42 – 49, 2004.

[30] Y. Yang, S. Tang, W. Zhang, and L. Fang, “A workflow language for grid services in ogsi-based grids,” *Lecture Notes in Computer Science*, vol. 3251, pp. 65 – 72, 2004.

[31] S. Krishnan, P. Wagstrom, and G. Laszewski. (2002) Gsfl: A workflow framework for grid services. [Online]. Available: http://users.sdsc.edu/~sriram/publications/gsfl.pdf

[32] A. Slomiski, “On using bpel extensibility to implement ogsi and wsrf grid workflows,” in *GGF10 Grid Work Flow Workshop*, 2004.

[33] C. Peltz, “Web services orchestration and choreogrphy,” *Web Services Journal*, vol. 03, no. 07, 2004.

[34] A. Kleppe, J. Warmer, and W. Bast, *MDA Explained.* Addison Wesley, 2003.

[35] (2005) Mof 2.0 query/view/transformation home page. [Online]. Available: http://www.omg.org/techprocess/meetings/schedule/MOF_2.0_Query_View_Transf._RFP.html

[36] (2005) Umt-qvt homepage. [Online]. Available: http://umt-qvt.sourceforge.net/

[37] OMG. (2004) Mof 2.0 core final adopted specification. [Online]. Available: http://www.omg.org/cgi-bin/doc?ptc/2003-10-04

[38] ——. (2002) Xml metadata interchange (xmi), v2.0. [Online]. Available: http://www.omg.org/cgi-bin/doc?formal/2003-05-02

[39] S. D, G. R, and S. I, “Web service composition in uml,” in *The 8th International IEEE Enterprise Distributed Object Computing Conference (EDOC)*, Monterey, California, 2004.

[40] T. Gardner. (2004) Business process definition metamodel. [Online]. Available: http://www.omg.org/cgi-bin/doc?bei/04-08-03

[41] ETH-Zurich. (2005) Jopera service composition suite homepage. [Online]. Available: http://www.iks.ethz.ch/jopera

[42] D. Box, E. Christensen, F. Curbera, D. Ferguson, J. Frey, M. Hadley, C. Kaler, D. Langworthy, F. Leymann, B. Lovering, S. Lucco, S. Millet, N. Mukhi, M. Nottingham, D. Orchard, J. Shewchuk, E. Sindambiwe, T. Storey, S. Weerawarana, and S. Winkler. (2004) Web services addressing (ws-addressing). [Online]. Available: http://www.w3.org/Submission/ws-addressing/

[43] J. Clark and S. DeRose. (1999) Xml path language (xpath) version 1.0. [Online]. Available: http://www.w3.org/TR/xpath

[44] B. Verheecke, M. A. CibrÃ¡n, and V. Jonckers, "Aspect-oriented programming for dynamic web service monitoring and selection," in *European Conference on Web Services*, Erfurt, 2004.

[45] K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. (2004) The ws-resource framework. [Online]. Available: http://www.globus.org/wsrf/specs/ws-wsrf.pdf

[46] S. Tuecke, L. Liu, and S. Meder. (2004) Web services base faults 1.2. [Online]. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[47] T. Maguire and D. Snelling. (2004) Web services service group 1.2. [Online]. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[48] S. Graham and J. Treadwell. (2004) Web service resource properties 1.2. [Online]. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[49] L. Srinivasan and T. Banks. (2004) Web services resource lifetime 1.2. [Online]. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[50] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke. (2004) Publish-subscribe notification for web services. [Online]. Available: www.ibm.com/developerworks/library/ws-pubsub

[51] J.-J. Dubray. (2005) Automata, state, actions, and interactions. [Online]. Available: http://www.ebpml.org/pi-calculus.htm

[52] D. Martin. (2005) Owl-s: Semantic markup for web services 1.1. [Online]. Available: http://www.daml.org/services/owl-s/1.1/overview/

[53] (2005) Web service modeling ontolgy home page. [Online]. Available: http://www.wsmo.org/

[54] (2005) Web service modeling language home page. [Online]. Available: http://www.wsmo.org/wsml/index.html

[55] A. Gramm. (2005) Ws-qos - a framework for qos-aware web services. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-tech/wsqos/tech-reports/FUB-Tech-Report_B-04-11_Gramm.pdf

[56] (2004) Uml profile for modeling quality of service and fault tolerance characteristics and mechanisms. [Online]. Available: http://www.omg.org/docs/ptc/04-09-01.pdf

[57] (1999) Xsl transformations. [Online]. Available: http://www.w3.org/TR/xslt

[58] W. v. d. Aalst. (2005) Workflow patterns home page. [Online]. Available: http://tmitwww.tm.tue.nl/research/patterns/patterns.htm

[59] W. M. P. v. d. Aalst, "Dont go with the flow: Web services composition standards exposed," *IEEE Intelligent System*, vol. 18, no. 1, pp. 72–85, 2003.

[60] W. M. P. v. d. Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros., "Workflow patterns," *Distributed and Parallel Databases*, vol. 14, no. 3, pp. 5–51, 2003.

[61] S. A. White. Process modeling notations and workflow patterns. [Online]. Available: http://www.bpmn.org/Documents/NotationsandWorkflowPatterns.pdf

[62] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual, Second Edition.* Addison-Wesley, 2004.

[63] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1988.

[64] Oracle. (2005) Oracle bpel process manager homepage. [Online]. Available: http://www.oracle.com/technology/products/ias/bpel/index.html

[65] IBM. (2004) Bpws4j homepage. [Online]. Available: http://www.alphaworks.ibm.com/tech/bpws4j

Appendix A

Presentation of Service Composition Languages

A.1 Presentation of existing visual service composition languages

Service composition is normally implemented as a specialized form of a executable business processes. This state of the art study will therefor focus on visual languages for defining business processes, but there will be specific focus on the support for service composition and heterogeneous service technologies. The languages will be presented from several viewpoints;

- External Aspects of the service composition
- Internal Aspects of the service composition
- Support for service composition
- Support for composition of heterogeneous services

. The languages presented are UML2, UML4EDOC (UML1.4 profile), BPMN, JOpera, and the BPDM notation.

A.1.1 UML2

The Unified Modeling Language [10],[62] was created 1997, and has been standardized and developed by OMG since then. Currently Uml2 is in the final stages of finalization. It has a explicit metamodel structure, MOF[37], which means that it is easily extendable, both at a metalevel and at a more concrete level. UML consists of several types of models, giving it an enhanced expressive power, being able to model from several complementing perspectives. This gives the ability to model both static and behavioural aspects of a system. In this section UML 2.0 will first be investigated, followed by some UML extensions, called UML profiles. UML has an explicitly defined metamodel and lexical representation, XMI [38].

This makes UML a well suited candidate for model transformation.
A UML profile is an extension of UML. Stereotypes are used to define specializations of the existing constructs in UML to give the modeler the possibility to model a specialized area i a more specialized way. The stereotypes can also be used together with a graphical notation to enhance the visual capabilities of modelling a special domain.

Externally visible structure There are several model views that can model externally visible structure in UML, mainly Component and Class Diagrams[10]. In the context of service composition and executable business processes component diagrams are the most interesting, with component being a more coarse grained structure, with ore focus on interfaces, both provided and required.
Externally component diagrams focuses on the interfaces that the component provides to other components and the interfaces it needs other components to provide. The interfaces can contain operations and attributes. This interfaces can be connected to the internal details of the component by the use of ports which are connection points for a structural component and its environment. The use of ports and required and provided interfaces can be seen in figure A.1. The figure shows a set of components that via ports provide some interfaces and require some others. These interfaces are then connected using the dependency association. Stereotyping are used on the ComposedService component to indicate that this component is a process.

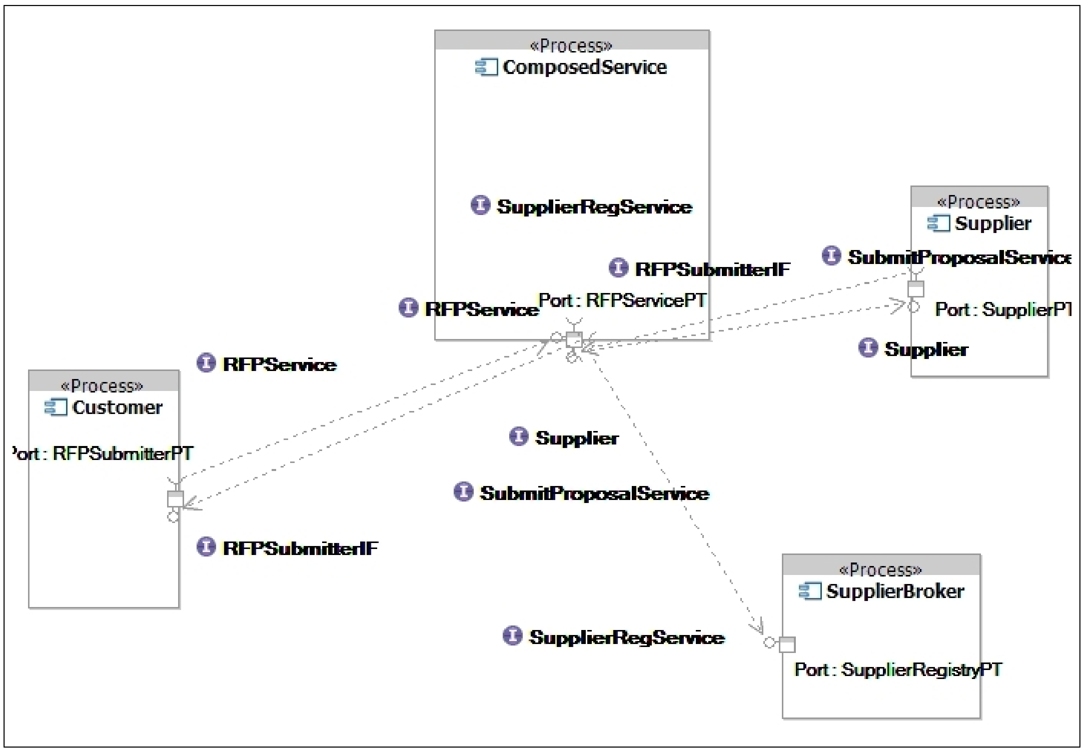


Figure A.1: UML2 Component Diagram

Internal structure The internal structure of a component is in UML 2.0 modeled using composite structure diagrams [10]. These diagrams show how the different parts of a component are composed and how they communicate. The parts can be connected to the ports of the components to create a connection to the external environment of the component. One can also use some parts of a component diagram to describe some internal detail, such as operation and attributes that are private and not externally visible.

Messages The messages that are part of a service composition or a process choreography can be defined using UML class diagrams [10]. The class diagrams takes an object oriented view to messages, giving a hierarchical structure, where classes contain attributes that themselves can be classes. Cardinality can also be expressed here giving the opportunity to use collections. For event and asynchronous messages a standardized stereotype «Signal» is possible to use. A class diagram is shown in figure A.2. Some classes are set to be parts of other classes by the use of the aggregation association.

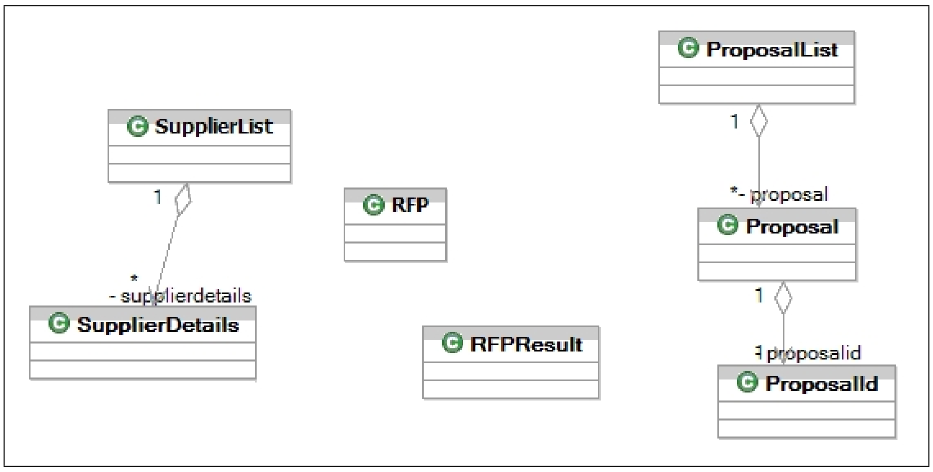


Figure A.2: UML2 Class Diagram

Externally visible behaviour The behaviour of a composed service or executable business process have several aspects. If one wants to show how the components, or services or processes, interact, only the externally visible behaviour is interesting, called the protocol of the interaction.
In UML 2.0 the external behaviour is described using interaction diagrams [10], which is either sequence or communication diagrams. These diagrams shows the possible histories of the messages being sent between the components, modelled as timeliness. The diagrams have programmatic constructs such as conditions ans loops called fragments. Interaction diagrams can be hierarchical using gates and diagram references to as constructs to connects together different levels in the hierarchy. Lifelines are used to model the participants in the diagrams interchanging messages.

The strength of sequence diagrams are that they support the concept of time, using a happened-before relation, where one can model that some event, such as sending or receiving a message must happen before another event in the same timeline and also that the sending of message from one component must happen before the message is received at the destination. A sequence diagram is shown in figure A.3. The lifelines represents the ports modelled in figure A.1, and there is one example of a fragment showing two alternative behaviours. The flow of dif-

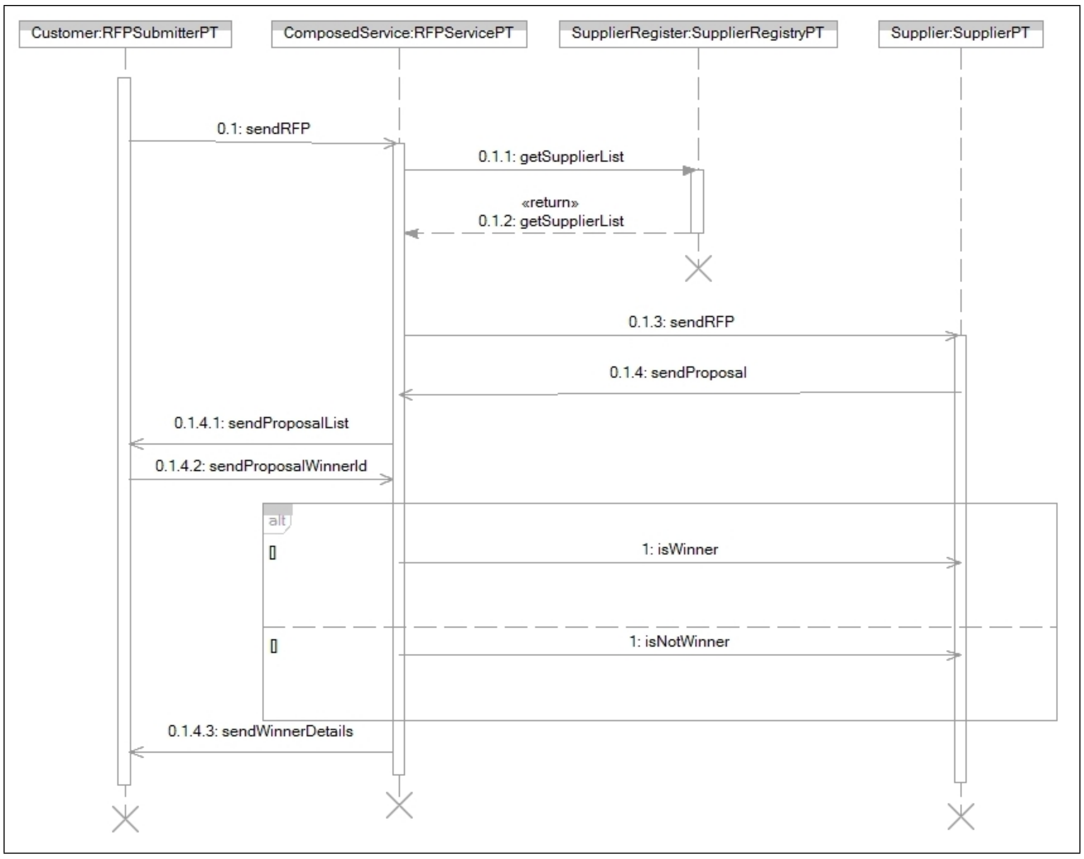


Figure A.3: UML2 Sequence diagram

ferent interactions can be shown by using an interaction overview diagrams, using concepts from activity diagrams to easily model parallel behaviour and decisions. Another model view that has the same expressive power as a sequence diagram is the communication diagrams. Third model view uses numbering of the messages being sent between the components instead of the implicit happened-before relation of the interactions. Semantically a communication diagrams is equal to an sequence diagram.

Internal behaviour Interaction does not show the internal behaviour of a component, a component can only send or receive a message. If the component wants to perform an internal operation , for instance changing the value of a variable, the interaction is not a sufficient model view. For this one should use

activity diagrams. This model view shows the flow of an process, both in terms of control flow, ie, what part of the system has control of the execution, and the dataflow, how the data flows between the different parts of the system. The graph of activities can be composed in a hierarchical fashion to create a better structure, using the fact that an activity can contain be decomposed into a set of activities.
At the atomic level, actions are the smallest unit of behaviour. The UML action language has a well defined semantics , making it as expressive as a lexical programming language. The are actions for sending and receiving messages, as well as calling operations, accepting calls and changing the values of variables. The actions contain the properties necessary to execute the defined behaviour. It can also have associated dataflow to show the input and output data for the action. Figure A.4 gives the internal behaviour of the ComposedService processcomponent. This process has a set of variable, given as «datastore» objects, and a set of actions and a controlflow. The expansion node indicates that data structured as a collection is coming in and that the expansion node loops over each element of the collection and performs the specified behaviour on each element in the collection.

Support for heterogeneous service composition. Even though UML2 has no specific notion of a service it can be modeled using components and interfaces. From a behavioural viewpoint consuming a service can be modeled as an action or by sending messages in a sequence diagram. As UML is platform independent the implementation technology of the service is not showing in a UML diagrams but one can create a profile which details this by using stereotyping.

A.1.2 UML 1.4 Profile for Development for Component Based Enterprise (EDOC)

UML4EDOC [13] is a UML extension that aims to simplify the development of component based enterprise systems by providing a platform independent model. It can be used both for business and system modelling, at different levels of granularity. There are several profiles giving different model views. All profiles have an explicit metamodel.

Component Collaboration Architecture Profile This profile gives a component view of the model. It has both external and internal viewpoints using ports as the connecting element. Firstly, the structure viewpoints models the components, with ports, and protocols gives a structural view of the external aspects of the components, including how the different components are connected and and how they interact. Also the interfaces of the components are defined in this view. An example of this diagram type is shown in figure A.5 where to process components communicate using a pair of ports, and the communication is defined in the protocol. . Next is the choreography viewpoint, figure A.6,

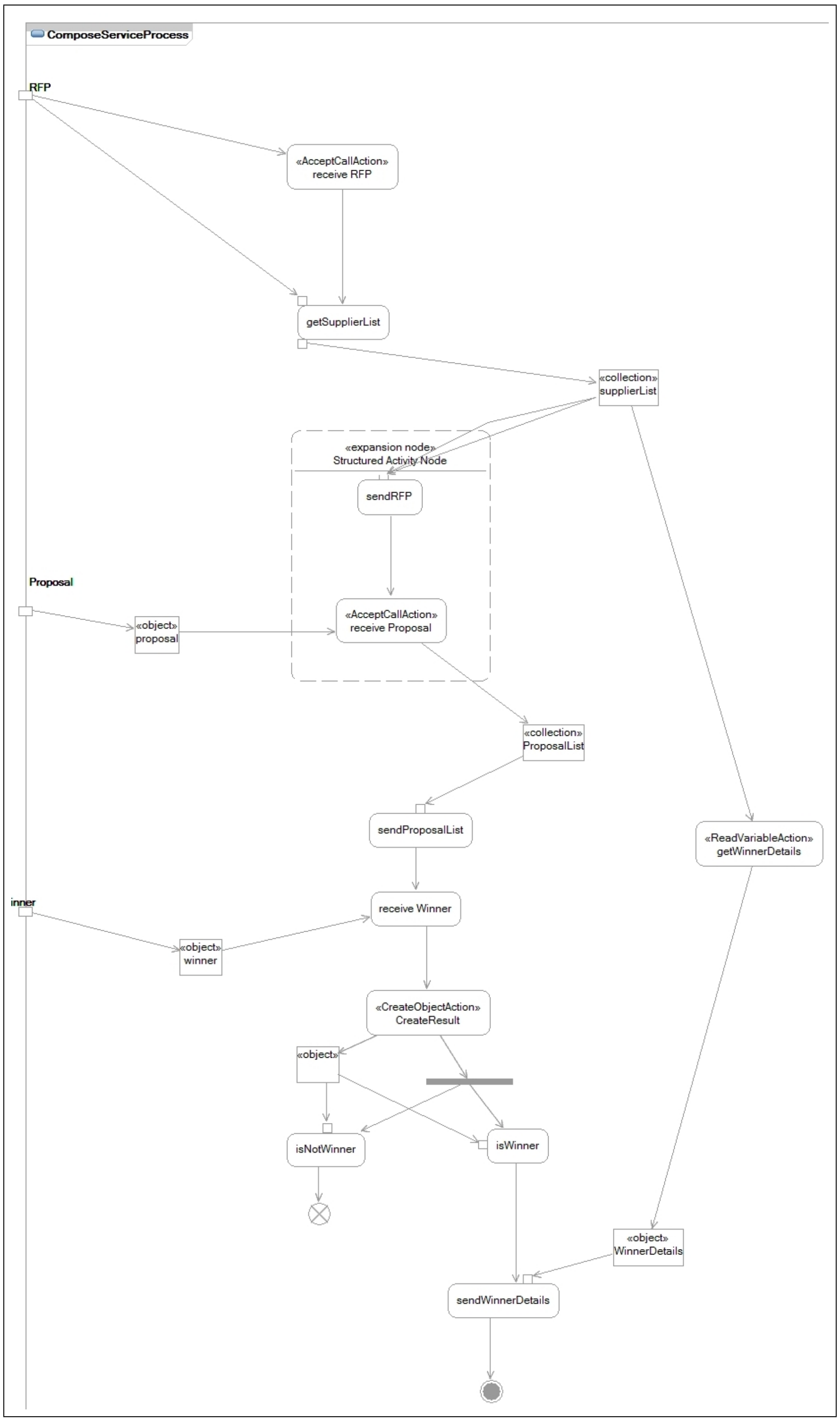

ComposeServiceProcess
RFP
«AcceptCallAction» receive RFP
getSupplierList
«collection» supplierList
«expansion node» Structured Activity Node
sendRFP
Proposal
«object» proposal
«AcceptCallAction» receive Proposal
«collection» ProposalList
sendProposalList
«ReadVariableAction» getWinnerDetails
inner
«object» winner
receive Winner
«CreateObjectAction» CreateResult
«object»
isNotWinner
isWinner
«object» WinnerDetails
sendWinnerDetails

Figure A.4: UML2 Activity diagram

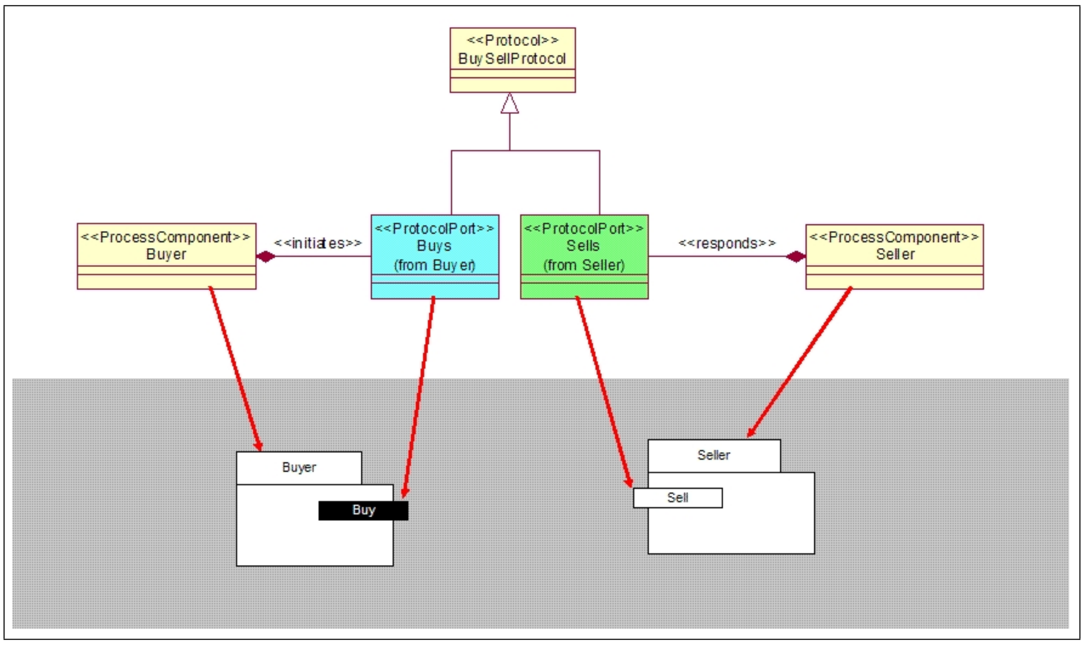


Figure A.5: UML4EDOC Structure Diagram

showing the behaviour of the component from an external point of view. . There is also a viewpoint in UML4EDOC that focuses on the internal structure of a processcomponent, and how the internal details are connected, via the ports, to the external environment.

Entities Profile The entities profile models the information view of the component. The structure of the dataobjects that are used and how they are associated with each other, and the aggregation is also shown. The entity model also introduces the notion of roles with the entity concepts. Entity roles are managers that provide ports that can be used to manage the data objects that are a part of the entity. Figure A.7 shows an entity diagram where an AccountManager role is associated with the entitydata and aggregation is used to define the parts of the entitydata object

Events Profile The events profile is a profile for modelling event base business process systems. With this profile one can show how the event are sent between processes, and what data they interact with. The publish/subscribe pattern is also supported by this profile.

Business Process Profile This profile focuses on the business process, and relating this process to the components. The profile is based on activity diagrams. It uses a different notation but most of the same concepts. The compundtasks can have ports, giving the possibility of defining the input and output data from a compundtask. A compound task contains activities and the dataflow between

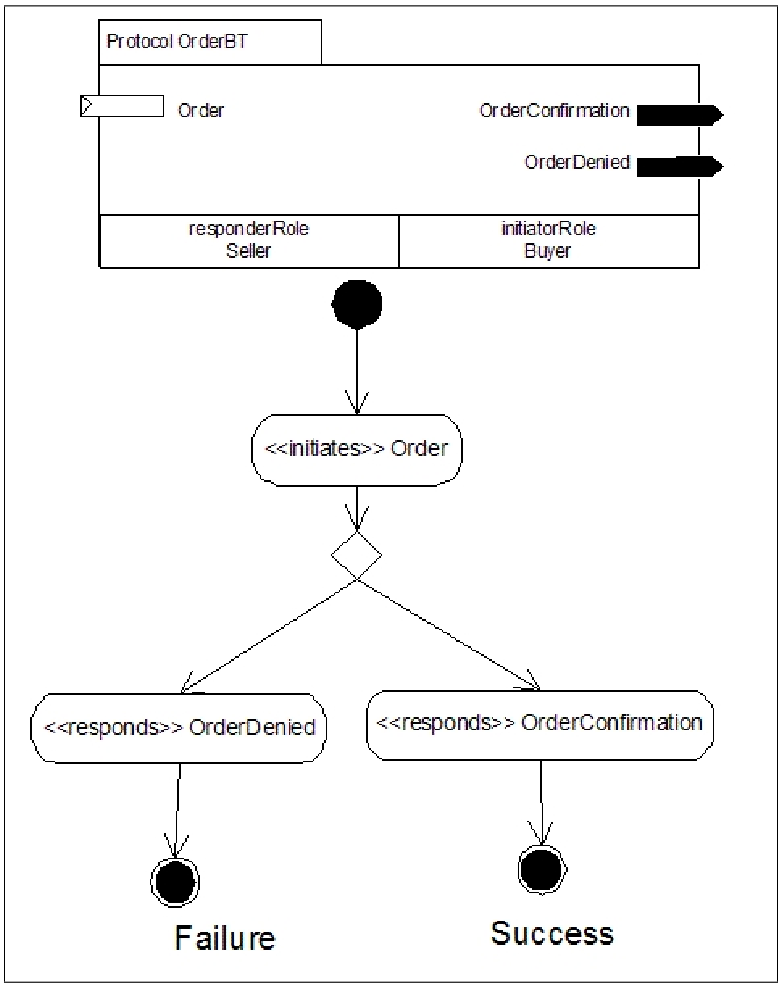


Figure A.6: UML4EDOC Choreography Diagram

them. Control flow is defined to be a specialized form of dataflow. Also in the business process view the notion of a role is introduced. A role is introduced as an abstraction of an external component that are used by the activities in the compound task. It can either be defined as a specific component or as a search-query used to find a component and bind to the role at runtime. This is used when an activity needs an component to do its work. There are three different roles defined, performer, artifact and responsible-party. The usage of roles is a way of connecting an activity with its environment, ie. the components that exists together with the process.
Figure A.8 shows a process that is defined using this profile. The processcomponents has ports, which is where the data is initially sent to the component. The activities A, B and C are similar to actions in UML2, representing a piece of work. In UML4EDOC these actions are associated with the role, for instance PR1, that performs the work. The dataflow is connected to the activities by using ports.

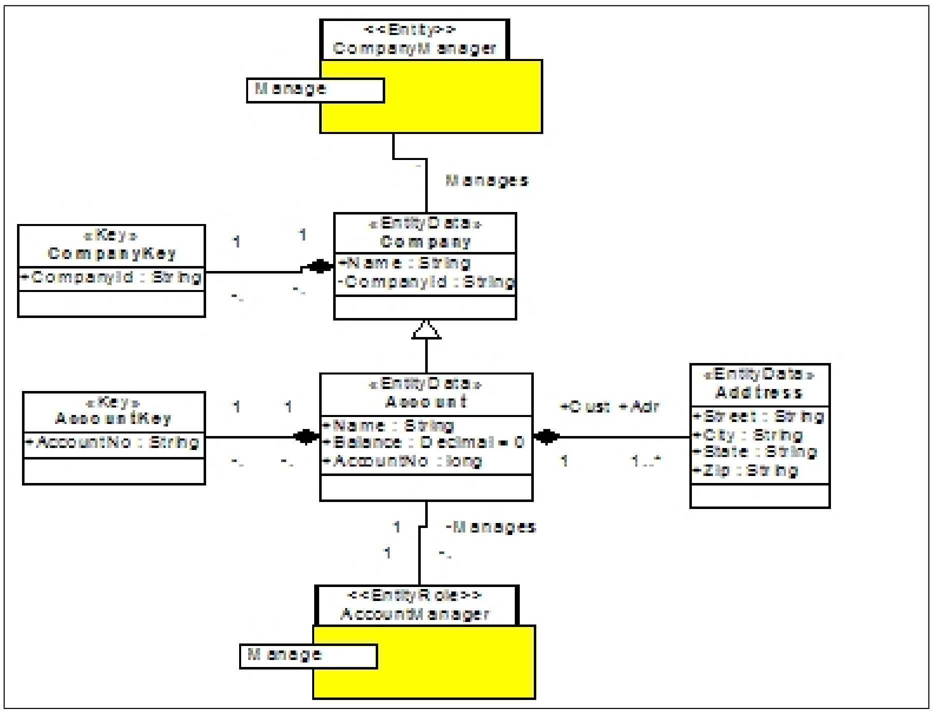


Figure A.7: UML4EDOC Entity Diagram

A.1.3 BPMN

The Business Process Modeling Notation [11] is developed by the BPMI.org organization. It is a visual language that should be readily understandable for business users. This makes it a different proposition from UML, which are much more software oriented in its approach. To achieve this, the BPMN notation models business processes at a high abstraction level which removes some of the details, and makes it easier to model the business aspects of the process, without considering the software point of view.
BPMN does only contain one model view, the business process diagram, and states in the specification that organization structure, functional breakdowns and data and information modelling is outside of the scope of BPMN.

Externally visible behaviour A process modelled in BPMN can be modelled to only focus on the external behaviour. The main units in such diagram is a pool. The pool is a part of the pool-swimlane metaphor which is used for creating structure in a Business Process Diagrams (BPD), with the pool being the main a separate process, which again can be divided into (swim)lanes to create structure inside a pool, for instance which roles or participants perform the specific activities. The pools can have dataflow, modelled as messages, between them. In these models the pools can be black boxes, giving no details about the internal behaviour. This model view does give some information about the choreography of the processes, but it does not give a complete model of the external

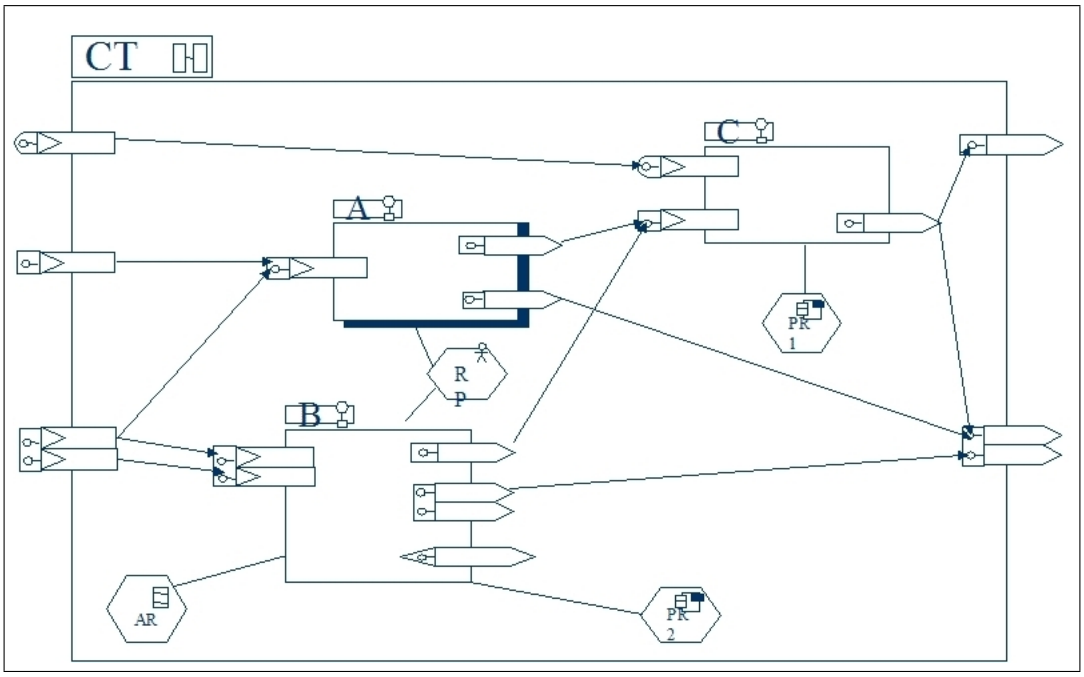


Figure A.8: UML4EDOC Business Process Diagram

behaviour. In the example shown in figure A.9, where the pools are partners in a process, and the dotted arrows between them are messages, one cannot tell if there are any time dependencies between the messages going between the different partners, such as the "RFP" message from the customer to the composed service and the "RFP Category" message going from the composed service to the supplier registry. The model is therefor not sufficient to show a protocol. It is also not possible in this view to show control logic such as loops of conditional behaviours.

Internal Behaviour Internally the BPMN notation contain three main types of objects, flow objects, connecting objects, swim lanes and artifacts. The flow object is either an event, an activity or gateways, while the connecting objects are sequence flow, message flow or association. The swim lanes can be pools or lanes. The specification states that the artifacts shall be a tool for the modeler to show extra information. Only three types of artifacts are specified, data object, group and annotation, but the modeler can add more if needed.
BPMN uses an graph based flow, with flow objects such as events activities or gateways being connected by connecting objects. The message flow can only go between pools, to model data flow between activities/events in one pool on e should use the dataobject artifact and association. An activity can also be a process, creating a hierarchy of processes that makes it possible to create more structure in the model.
The diagram in figure A.10 shows the internal behaviour of one of the partners from the example shown in figure A.9. This gives more detail, and shows the

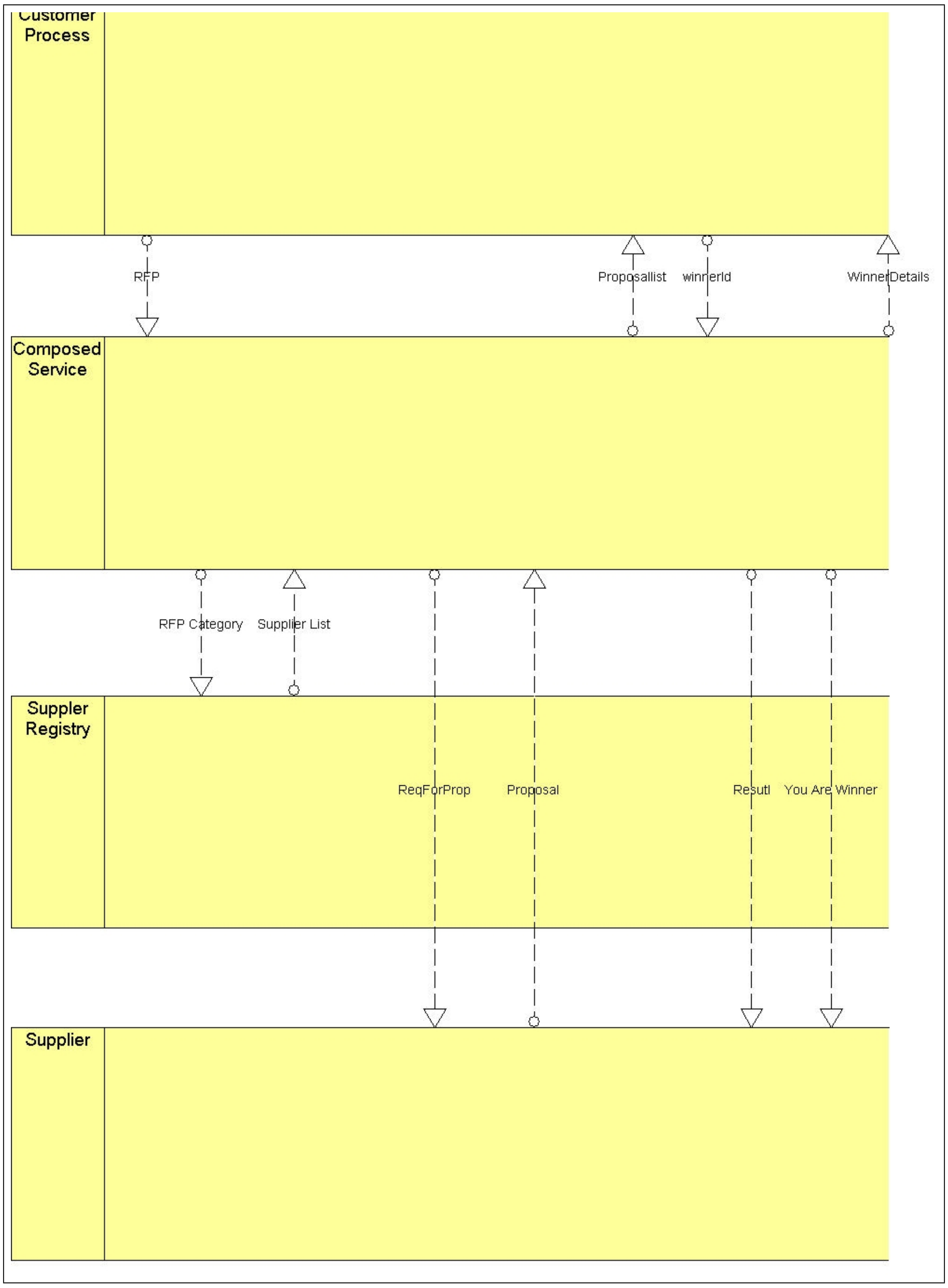


Figure A.9: BPMN External BehaviourDiagram

control flow inside a component and how it processes messages. Internal data handing is shown by using the data object artifact and associating this with tasks. There are also conditional logic in the form of a control gateway splitting the control flow of the process.

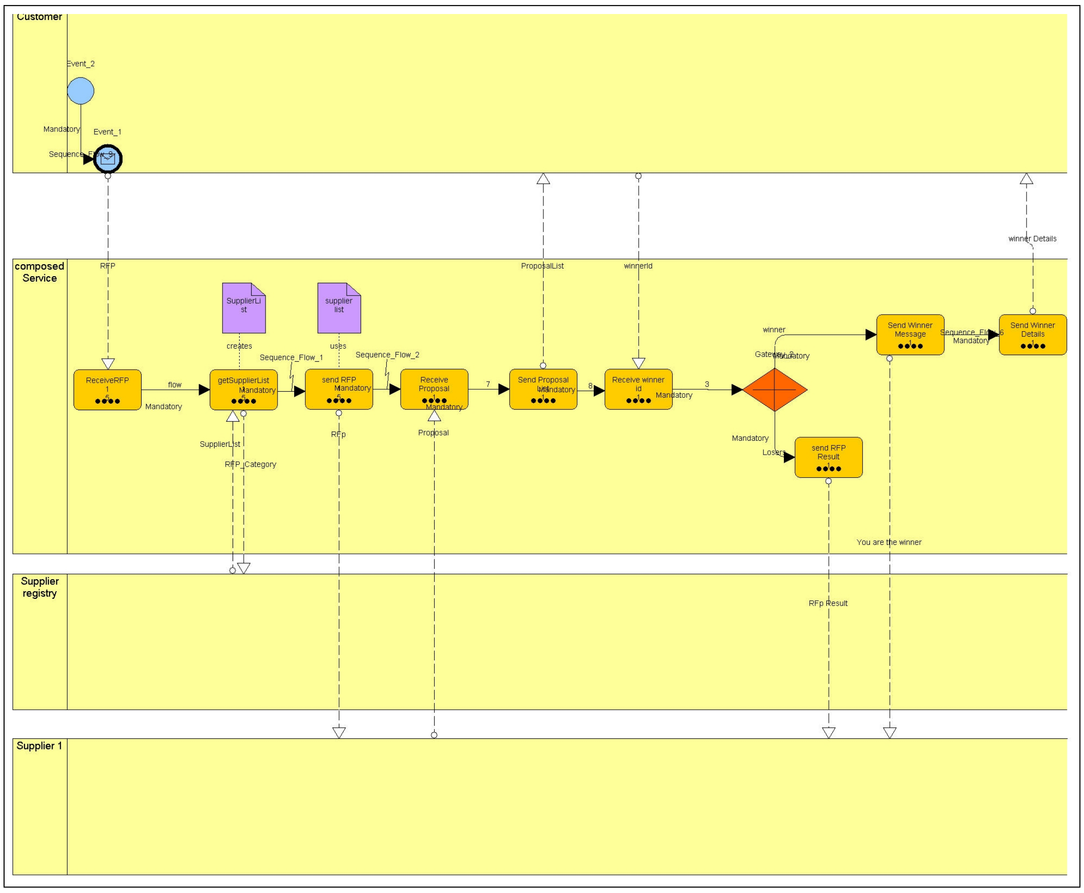


Figure A.10: BPMN Internal Behaviour Diagram

Heterogeneous Service composition A task in BPMN can be defined to be a service invocation. This is however specialized to Web Services, but the specification states that BPMN can be extended with other types of task-types thus making it possible to create service-invocation tasks for other service implementation technologies.

A.1.4 JOpera

Jopera is a [41] service composition suite, using a proprietary visual language, together with a lexical execution language. Here we will focus on the visual language. The language focuses on the internal view of the service composition, but is tightly integrated with the lexical language and the composition suite..

Internal View The model in JOpera is separated into two views, one for control flow and one for dataflow. There are no visual operators for control logic, theirs logis is hidden in the condition on activities, called programs, adding condition operators on connectors. Dataflow is shown as objects flowing between the programs, and transformations are performed in the flow, not as separate activities. There are noe visual way of indicating subprocesses, these are similar to external services. All details of an activity is represented textually, using the composition suite, the activity is just shown with a name, and the connected dataflow. Structure other than subprocess hierarchies are not possible. Figure

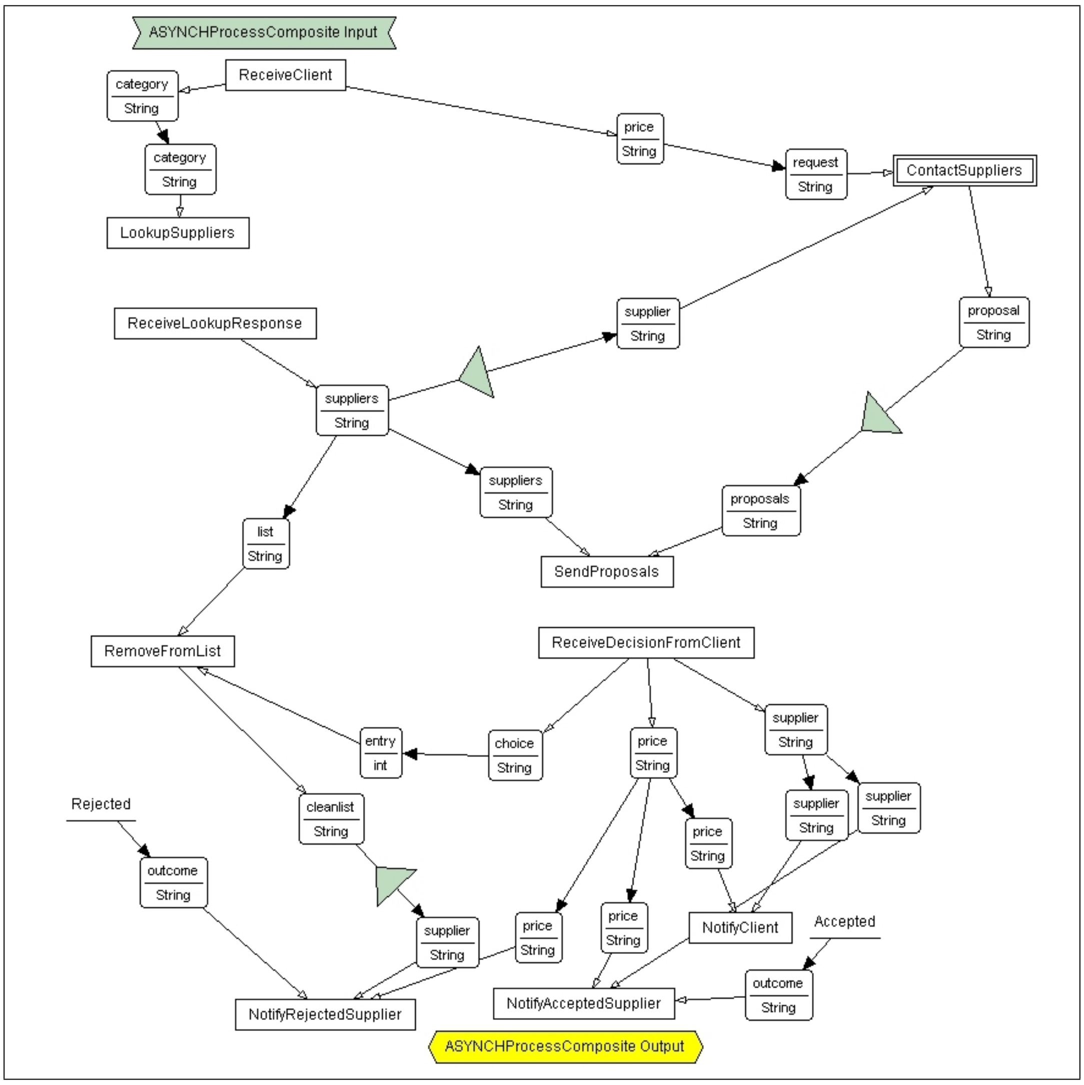


Figure A.11: JOpera Data Flow

A.11 shows the dataflow in a service composition example. The rectangles are activities, while the rounded rectangles are data objects. The flow objects with large arrows indicate relationships between collections and elements of that collection. Arrows with filled arrowheads are dataflows with transformation and arrows with open arrowheads are simple dataflow. Figure A.12 is a diagrams of the control flow of the same service composition that is shown in figure A.11. Again the rectangles indicate the activities and the dotted arrows indicate the controlflow.

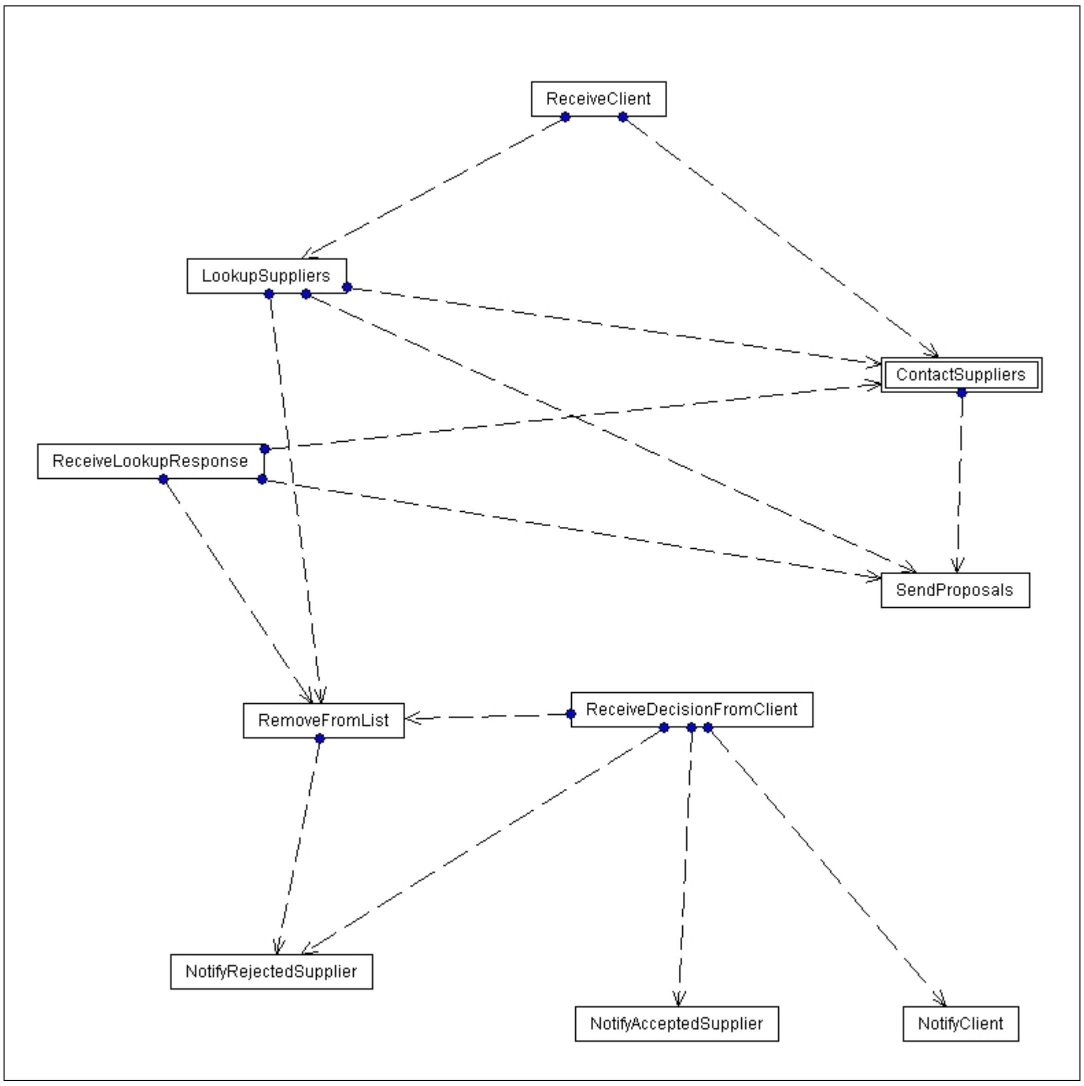


Figure A.12: JOpera Control Flow

Heterogeneous Service composition JOpera has support for many service implementation technologies, both for Web- and grid- services. The programs are services, either coarse grained or fine grained such a programs defined in Javascript.

A.1.5 BPDM notation

A notation for visually modelling business processes are used in the BPDM specification [40] and detailed in a separate white paper [12]. The specification states that this notation maintains consistency with the BPD-metamodel and its semantics. The modelling notation contains two main views, the internal and the external, showing how it is performed and how it interacts respectively. BPDM focuses on a business process perspective and modelling them in a complete way, using both behavioural and structural model views.

External View The external view uses different model views to show different levels of detail. The collaborative view, shown i figure A.13 models the collaborations that occur between partners, and defines the roles that the different partners must fulfill. The central business process can either be shown as a set of roles, or as just the process component. The protocol of the business process,

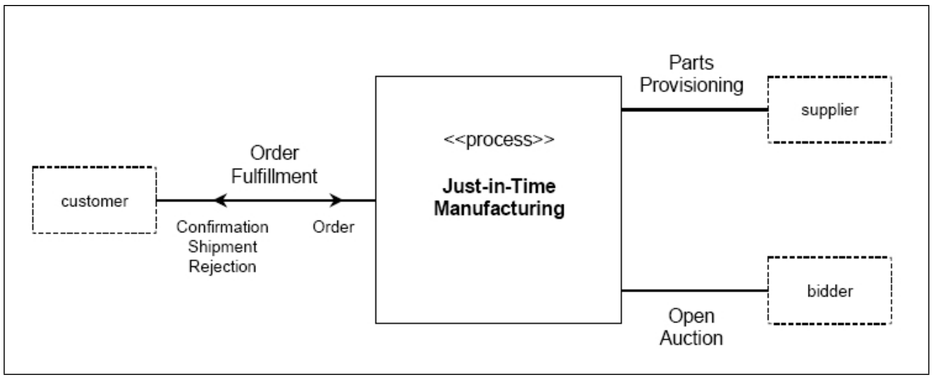


Figure A.13: BPDM Collaborations

which details the collaboration view with time information, in the sense of a notions of what activities must happen before others, and the different paths that can be taken through execution of the process. Figure A.14 shows such an protocol, showing the messages flowing between two roles. This diagram type can model control logis such as parallelism and conditions.

Internal View The internal view of this notation makes intensive use of concepts from UML2. Each piece of work is modelled as an task, which can be structured, ie. a composite, and a may have one or more exit and entry points. The tasks can have input and output, and some of these can be grouped together into sets, showing that only a specified subset of all input data must be available for the task to execute. The task can be human, automated, and may involve other partners. Figure A.15 shows such a process. The tasks have pins for inout and output data, and these pins can be grouped together to form sets of data. Activities that involve sending or receiving message have a slightly different notation than a manual task. Process variables are used as well. Some other aspects

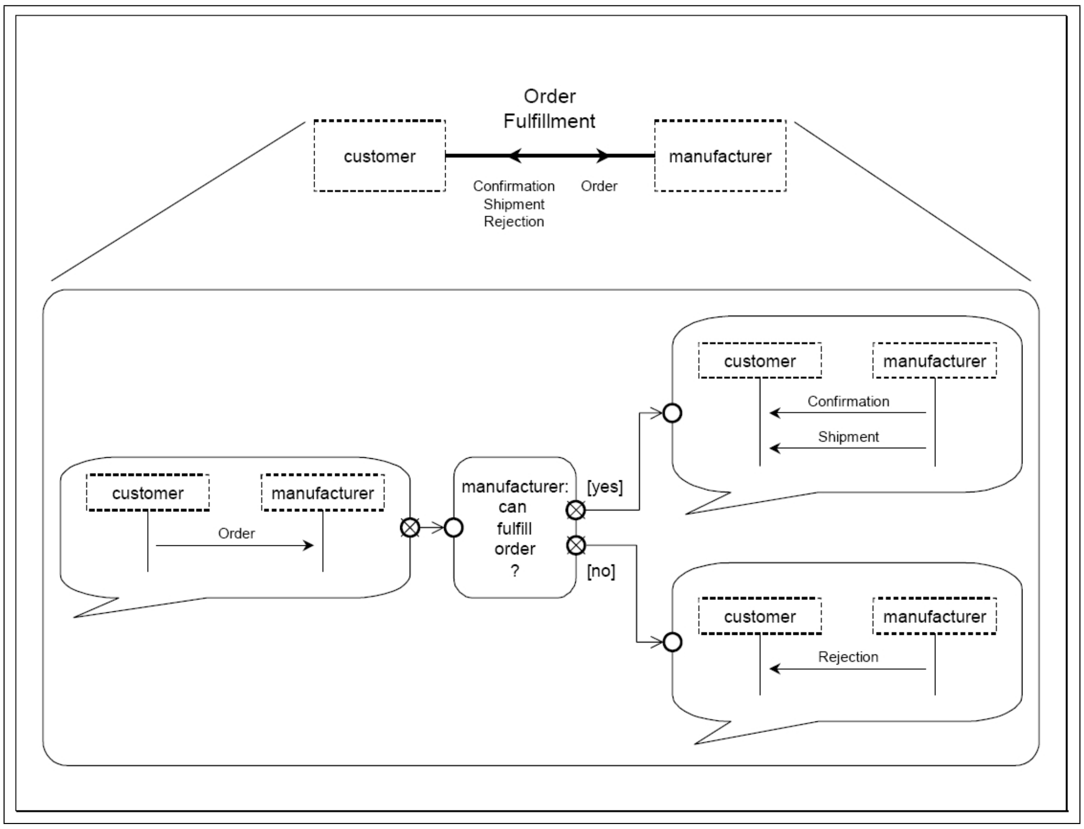


Figure A.14: BPDM Collaboration Protocol

with this notation, at least in this example, is the lack of flow control constructs, branches, forks and joins. Rather they use conditions on the flows going in and out of the tasks. They argue that such decisions take time in real life, and should as s uch be modelled as tasks. The example also uses repositories for storing data that are used by several tasks throughout the process. The repositories are used together with conventional dataflow. The notation also calls the repositories process variables. The modelling notation also contains a special kind of task for looping over a set of objects. There are specialized constructs in the notation for handling correlation information and dynamic binding of services to roles.

Heterogeneous service composition This notation focuses on business processes, does not contain any special notation of a service, but a task can be a service invocation, modelled as a interaction between roles passing messages to each other. the implementation technology of the service is not shown.

A.1.6 Proprietary Commercial Solutions

Several commercial products, mainly execution engines for business processes comes with some visual language for designing processes. These include [64] and [65], these are engines for executing BPEL [9]. The visual languages that are

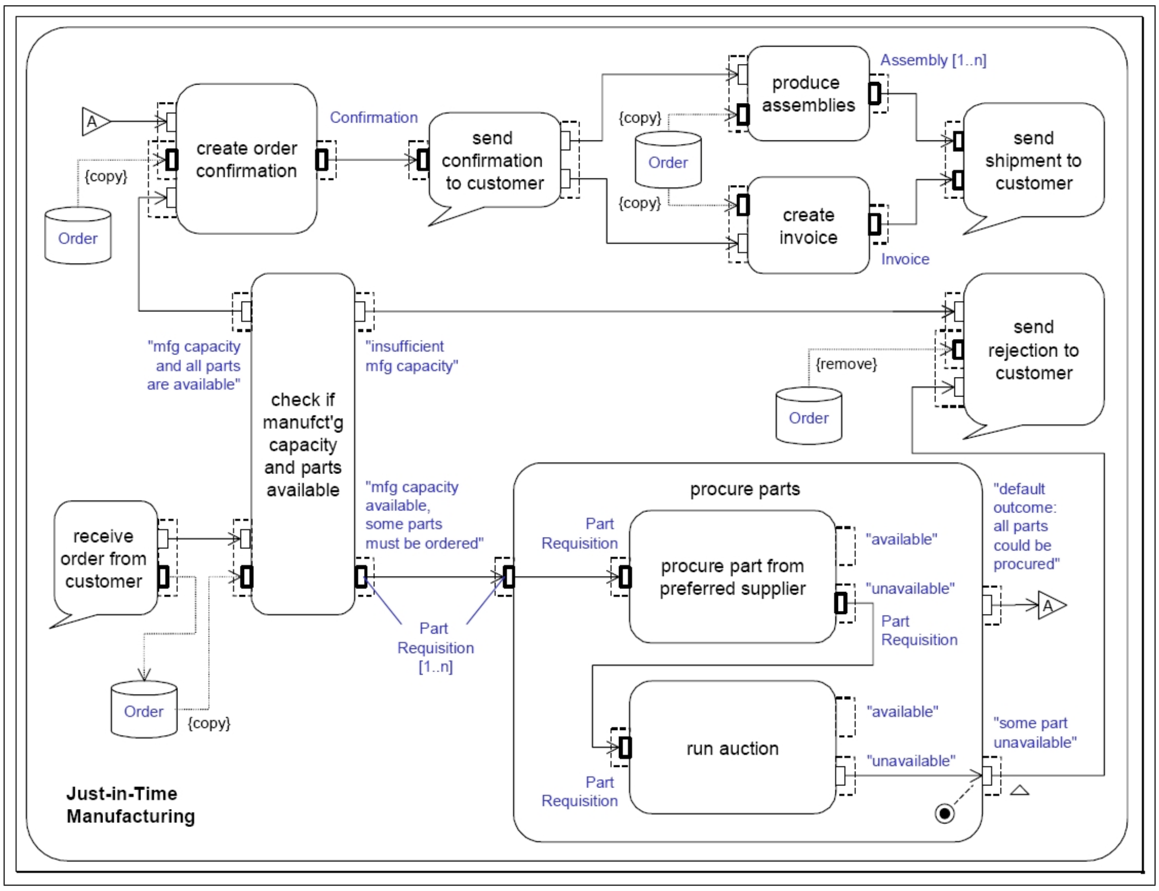

A
create order confirmation
{copy}
Order
Confirmation
send confirmation to customer
{copy}
Order
{copy}
produce assemblies
Assembly [1..n]
create invoice
Invoice
send shipment to customer
"mfg capacity and all parts are available"
check if manufct'g capacity and parts available
"insufficient mfg capacity"
send rejection to customer
{remove}
Order
"mfg capacity available, some parts must be ordered"
receive order from customer
Order
{copy}
Part Requisition [1..n]
procure parts
Part Requisition
procure part from preferred supplier
"available"
"unavailable"
Part Requisition
run auction
Part Requisition
"available"
"unavailable"
"default outcome: all parts could be procured"
A
"some part unavailable"
Just-in-Time Manufacturing

Figure A.15: BPDM Process

used are thus closely related to BPEL, and contains more or less a one-to-one mapping to the BPEL xml-schema. As these models is a the same abstraction levels as the execution language, they can be as more platform specific models than the modelling languages mentioned earlier.

Appendix B

Expressiveness - Workpatterns

B.1 Table of standards support

This table is from [58] and lists the direct support of workpatterns for UML and BPEL. Results from [61] have been added to show BPMN's support for these workpatterns.
[58] states that a pattern is only supported directly if there is a feature provided by the language which supports the construct without resorting to any of solutions mentioned in the implementation part of the pattern.
[61] compares UML activity diagrams with BPMN, with respect to mentioned workpatterns. [61] shows that even though some patterns are not directly supported, they are easily supported by combining constructs in the language. Thus [61] comes to different conclusions with respect to UML. These difference is shown as * in the table, indicating that even though [58] says that UML does not support this specific pattern, [61] shows that the pattern is supported.

Workpattern	BPEL	UML	BPMN
Sequence	+	+	+
Parallel Split	+	+	+
Synchronization	+	+	+
Exclusive Choice	+	+	+
Simple Merge	+	+	+
Multi Choice	-	- *	+
Synchronizing Merge	+	- *	+
Multi Merge	-	- *	+
Discriminator	-	-	+
Arbitrary Cycles	-	- *	+
Implicit Termination	+	- *	+
MI without Synchronization	+	-	+
MI with a Priori Design Time Knowledge	-	+	+
MI with a Priori Runtime Knowledge	-	+	+
MI without a Priori Run-time Knowledge	-	- *	+
Deferred Choice	+	+	
Interleaved Parallel Routing	+/-	- *	+
Milestone	-	- *	+
Cancel Activity	+	+	+
Cancel Case	+	+	+

B.1.1 5 basic Workpatterns

This section presents the 5 basic workpatterns as defined by [61].

B.1.1.1 Sequence

An activity in a workflow process is enabled after the completion of another activity in the same process [58]. See figure B.1

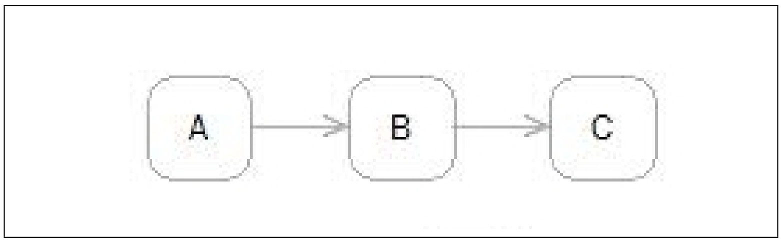


Figure B.1: Sequence workpattern

B.1.1.2 Parallel Split

A point in the workflow process where a single thread of control splits into multiple threads of control which can be executed in parallel, thus allowing activities to be executed simultaneously or in any order [58]. See figure B.2

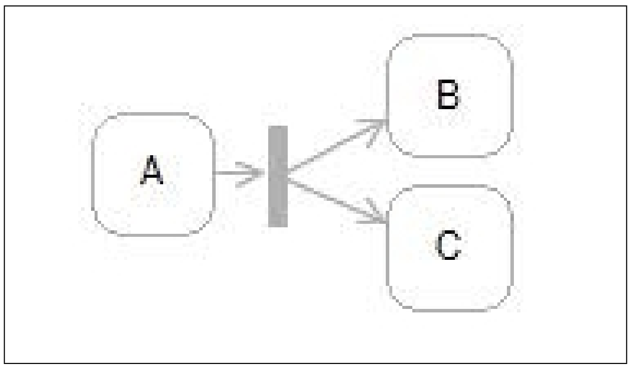


Figure B.2: Parallel Split workpattern

B.1.1.3 Synchronization

A point in the workflow process where multiple parallel subprocesses/activities converge into one single thread of control, thus synchronizing multiple threads. It is an assumption of this pattern that each incoming branch of a synchronizer is executed only once [58]. See figure B.3

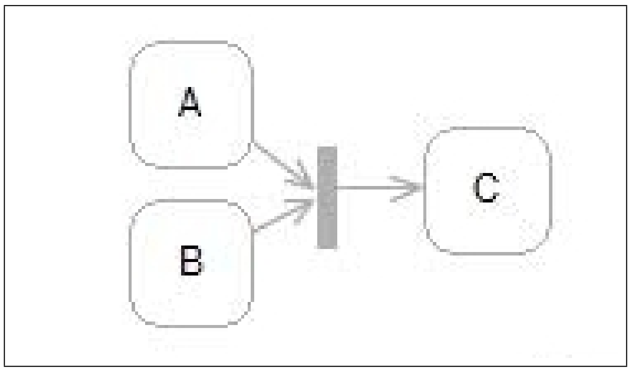


Figure B.3: Synchronization workpattern

B.1.1.4 Exclusive Choice

A point in the workflow process where multiple parallel subprocesses/activities converge into one single thread of control, thus synchronizing multiple threads. It is an assumption of this pattern that each incoming branch of a synchronizer is executed only once [58]. See figure B.4

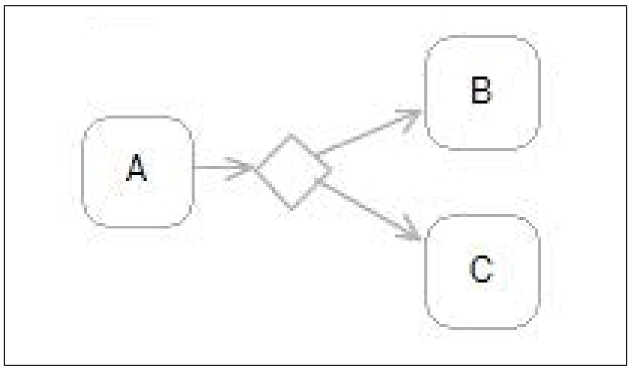


Figure B.4: Exclusive Choice workpattern

B.1.1.5 Simple Merge

A point in the workflow process where two or more alternative branches come together without synchronization. It is an assumption of this pattern that none of the alternative branches is ever executed in parallel [58]. See figure B.5

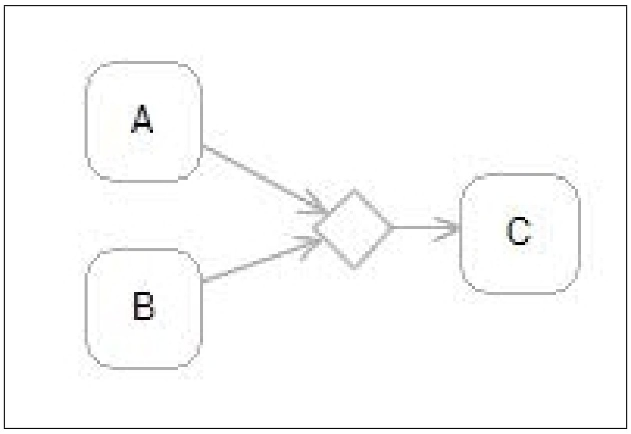


Figure B.5: Simple Merge workpattern

Appendix C

Case A - Request For Proposal

C.1 Case Description

This is a case based on conversation between several partners. The goal of the service is to get proposals from several suppliers from based on a "Request for Proposal" from the client. A use case diagram of the case can be seen in figure C.1

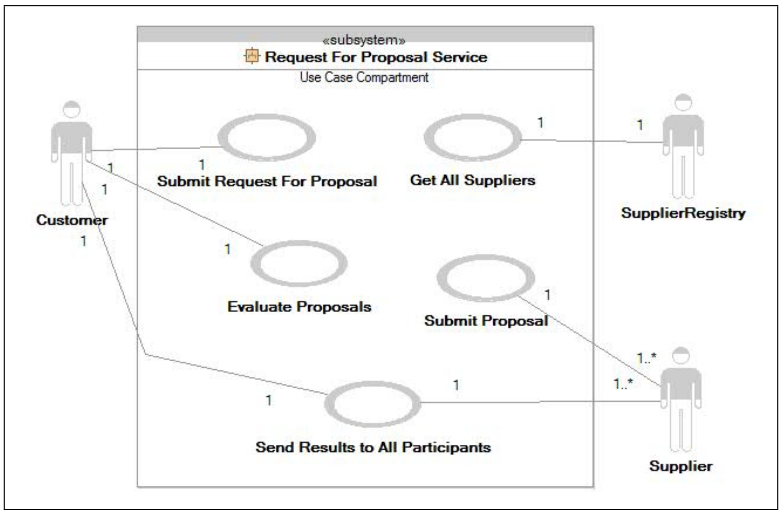


Figure C.1: UML2 Use case description of the RFP case.

Actors

- Customer - The sender of the Request for proposal. The customer must be able to receive messages from the RFP service as well as send messages.
- RFPService - The service which orchestrates the consumption of the other services. This actor orchestrates the service interactions with all other actors.

- SupplierRegistry - A yellow-pages type service.
- Supplier - A supplier that can perform the work described in the request for proposal. This actor must receive messages from the RFPService and also send messages back.

Behaviour First a Request for proposal is sent from the customer to the RFPService. Based on the contents of this message, the RFPService sends a query to the SupplierRegistry to get information of all appropriate suppliers. When the RFPService gets this list, the request for proposal is sent to all suppliers. The supplier sends a proposal back to the RFPService, which collects all proposals in a list, and sends the list to the customer. The customer sends back its decision of which proposal is the best. The RFP service then sends this result to all suppliers that submitted a proposal, and send information about the winning supplier to the customer.

Issues

- Conversations - There are conversational interactions between the Customer and the RFPService and the RFPService and the suppliers.
- Asynchronous Communication - The communication between the RFP service and its partner are asynchronous.
- Partner-requirements. The customer that wants to invoke the RFPService must have implemented the interfaces that are necessary to receive the defined messages which are a part of the expected behaviour.
- Dynamic service selection. The RFPService does not know the details of the suppliers before it is received from the SupplierRegistry. RFPService must handle a list of service-endpoint definitions and at runtime use this information to invoke the services.

C.2 UML2 Service Composition

This section contains an implementation of the Request for proposal case, using UML2. The UML2 models contains a class diagram specifying the interfaces and classes used in the model, a component diagram used to define the which interfaces are implemented by which components, and a sequence diagram and an activity diagram to model the behaviour
Figure C.2 is the UML2 class diagrams modelling the interfaces and classes used in this case implementation. The interfaces show the interfaces that the partner must implement, and the classes are the classes used in the interface for communication.

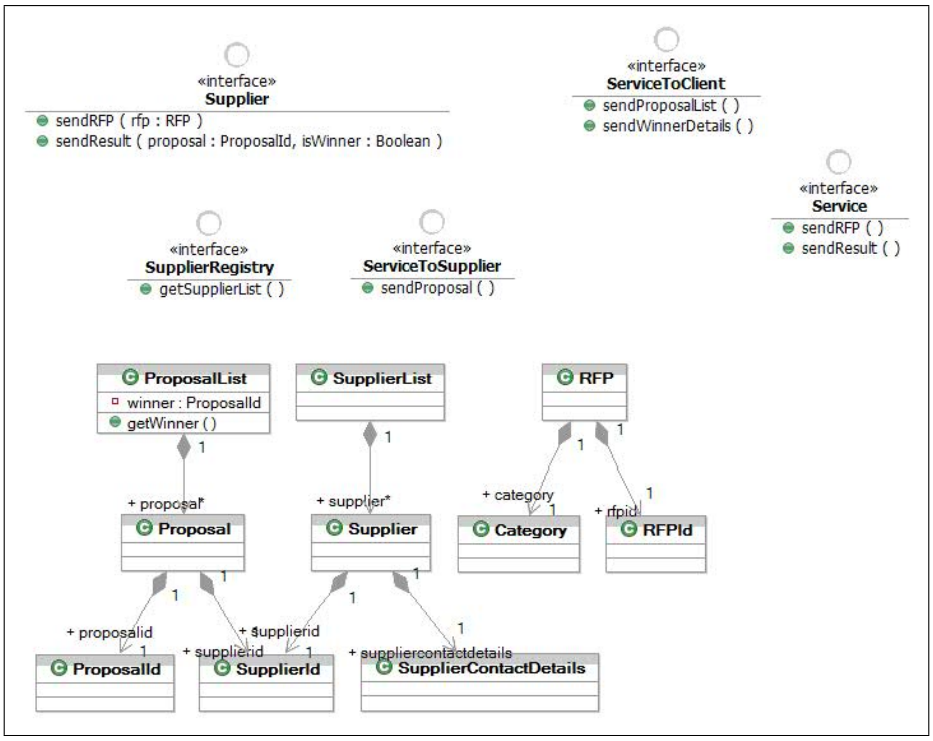


Figure C.2: UML2 Interfaces and Classes.

Figure C.3 uses the interface defined in figure C.2, and specifies the components that implements these interfaces. The components both provide interfaces to other components and specify what interface they require from other components. Required and provided interfaces are shown with the ball/socket notation on the components. Dependecies are introduced to connect the required interface of one components with an provided interface of another component.

Figure C.4 shows the interaction between the components from figure C.3.

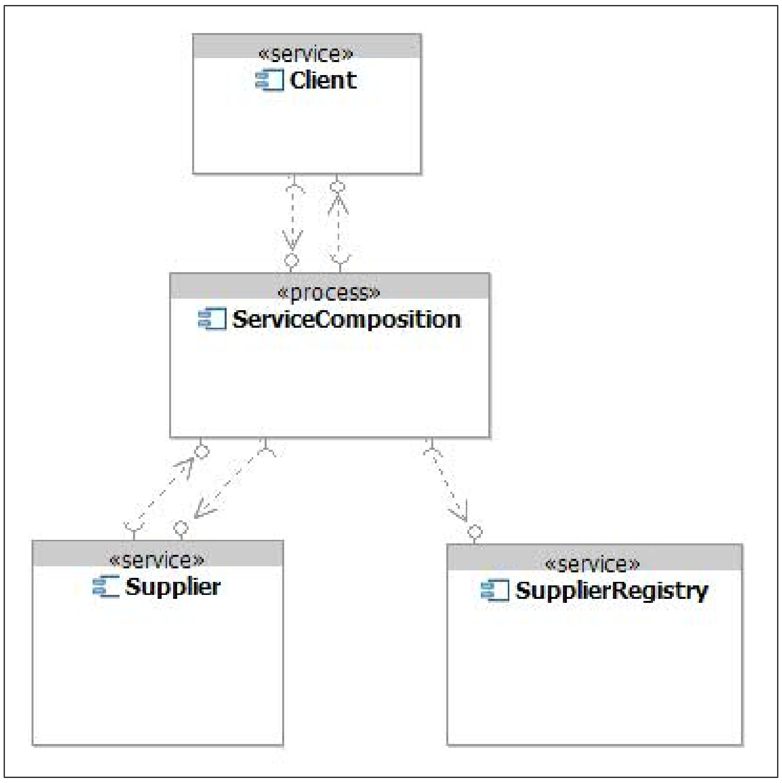


Figure C.3: UML2 Component Dependencies.

The sequence diagrams shows the message exchanges that er performed throughout the execution of the service composition. The model view uses the complex features in sequence diagrams such as loops and alternative behaviours.

Figure C.5 is the internal process of the service composition. The expansion node is a construct for looping over collection of data. In the first case the supplierlist is looped over, and for each element in the list, the actions inside the expansion node is performed. Each action represent either sending or receiving a message.

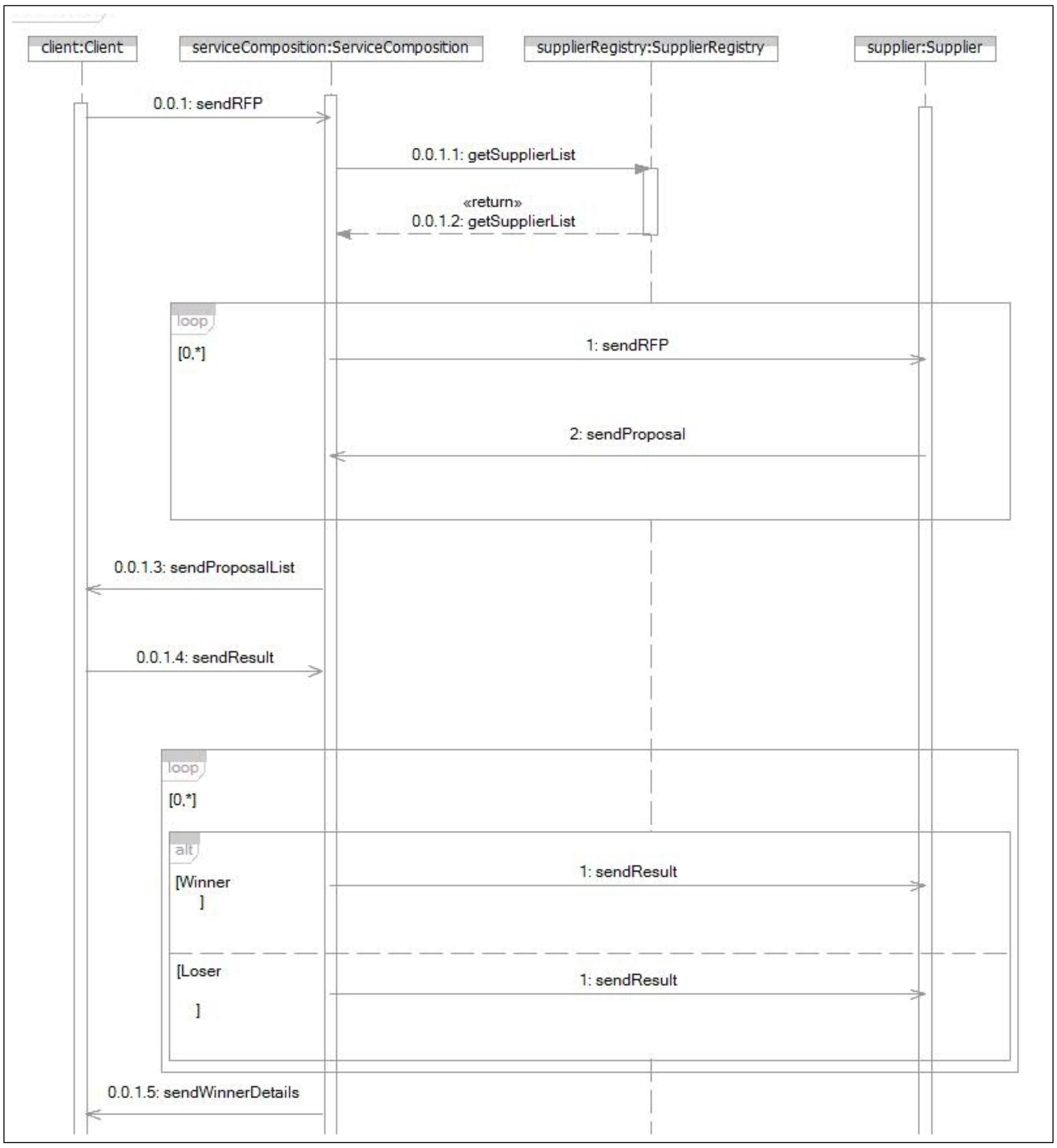


Figure C.4: UML2 External Protocol.

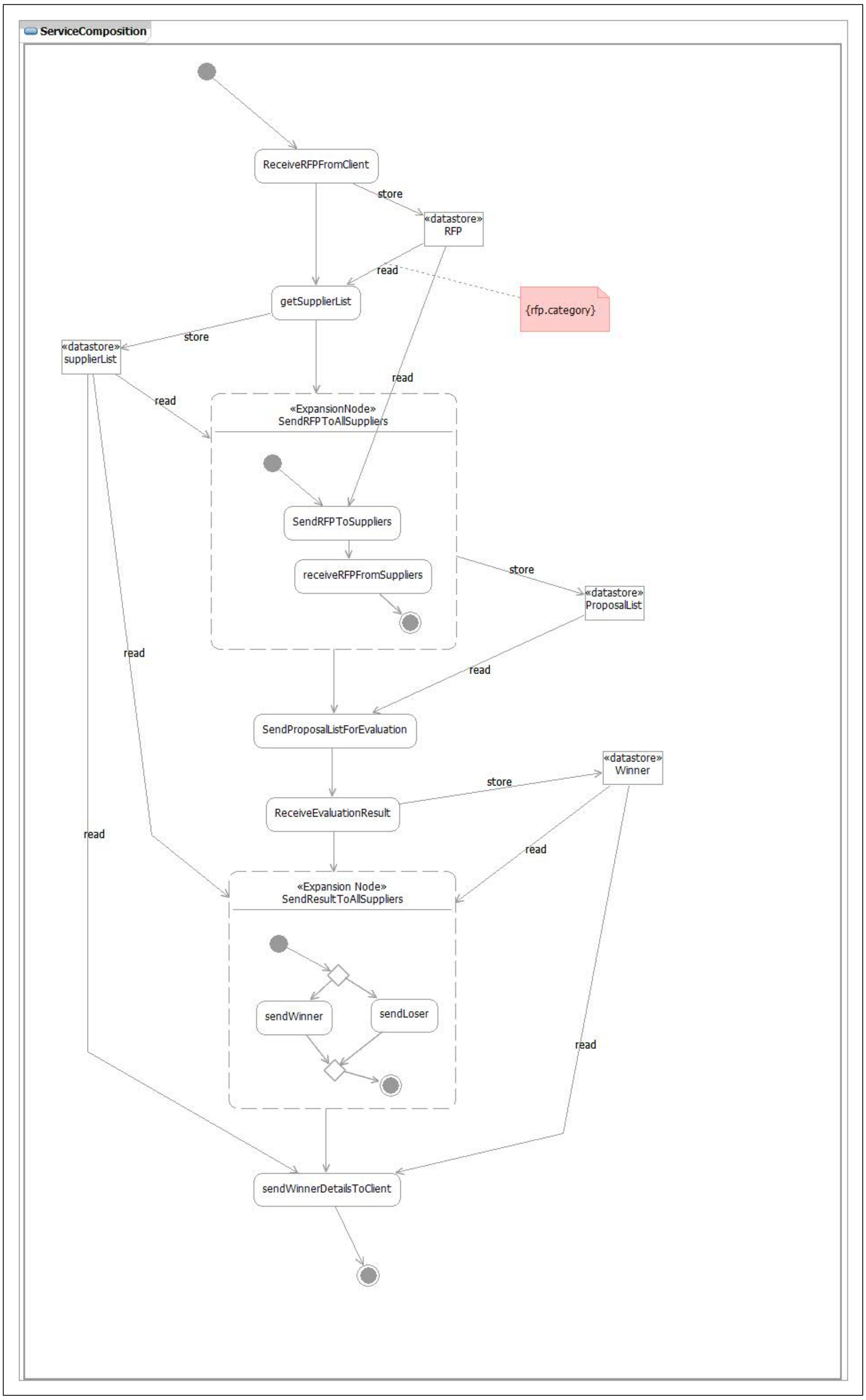


Figure C.5: UML2 Internal Process.

C.3 BPMN

In this section an implementation of the RFP case in BPMN is presented. The first figure shows the high level view, with no details of the internal details of the processes. The second model shows the internal details of the service composition.
Figure C.6 shows the RFP case from an external viewpoint, focusing on the message interchanges between the partners. The partners are shown as pools, and the model does not reveal any internal details.

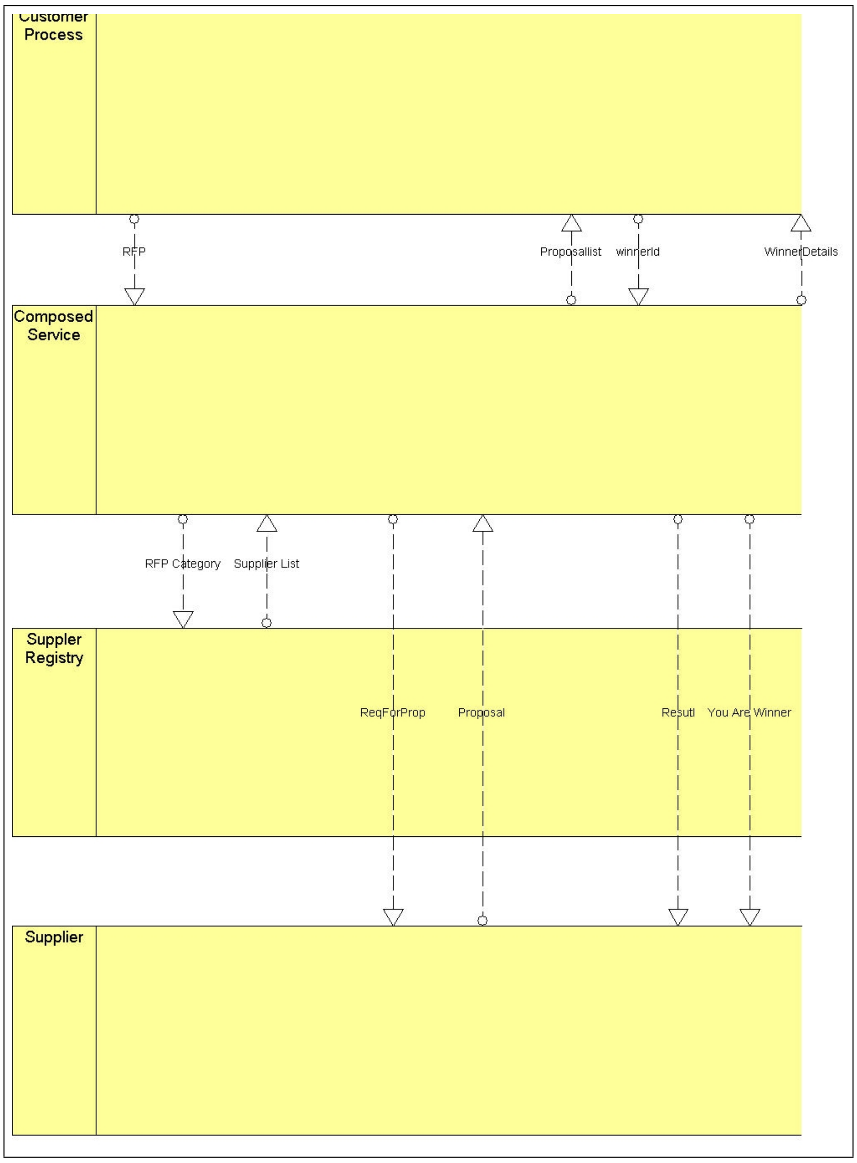


Figure C.6: Business Process Modelling Notation External Model.

Figure C.7 details the behaviour of the RFP service composition. The service communicates with other partners by sending messages, shown as message flows in the diagram. The internal control flow is also shown. Each tasks is a service service interaction. This model reveals details of the internal behaviour of the process.

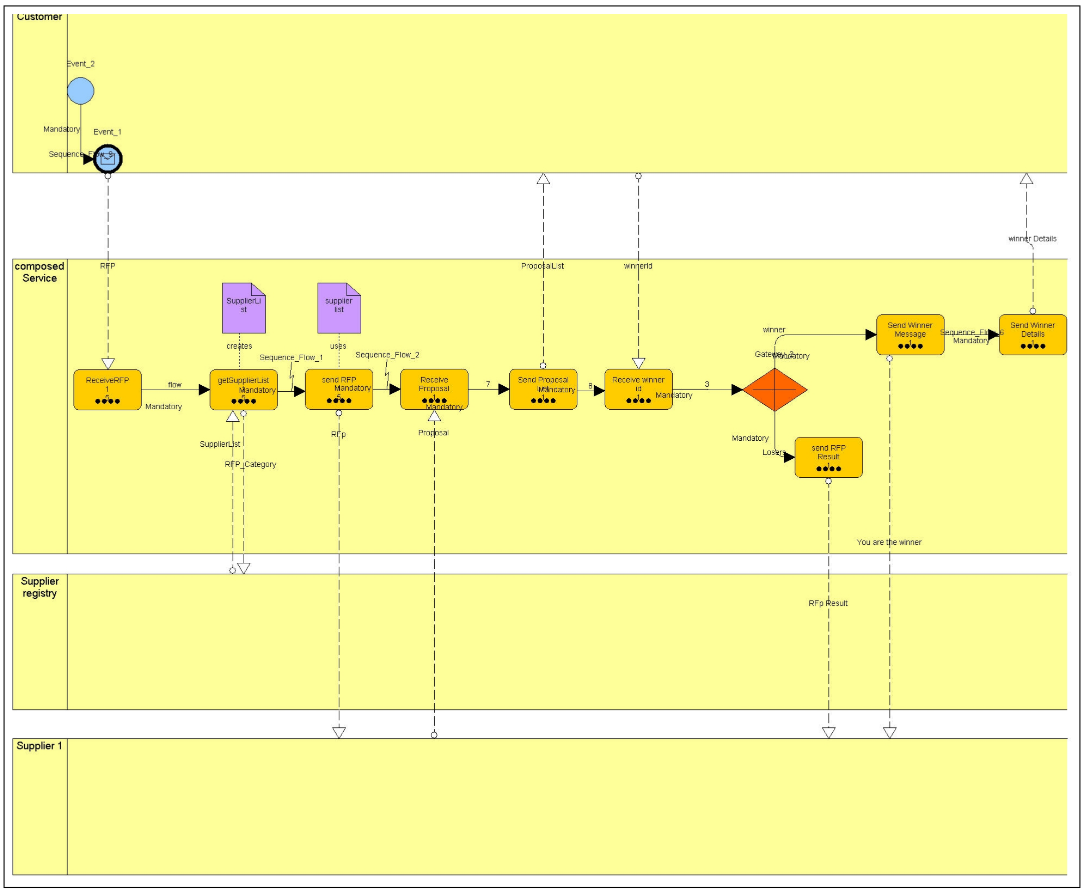


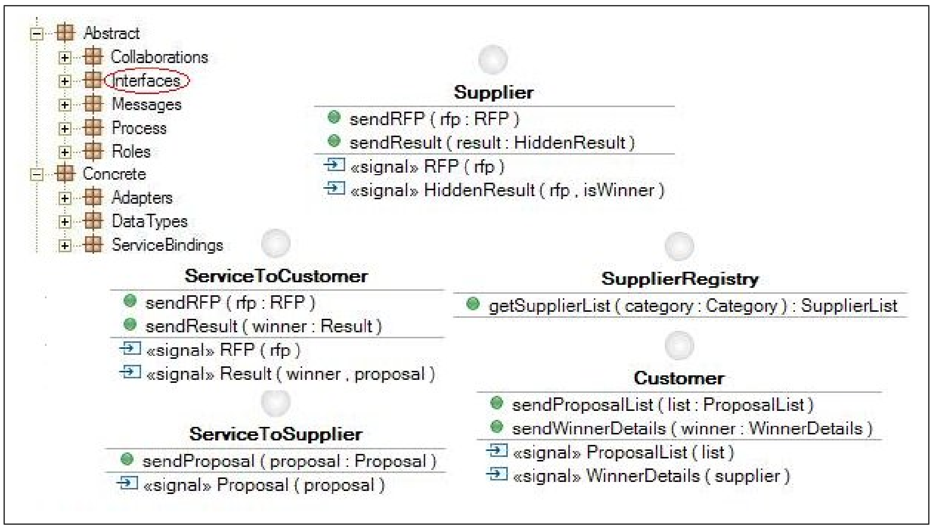
Figure C.7: Business Process Modelling Notation Internal Model.

C.4 AuSCL

This section presents a implementation of the RFP case using the AuSCL UML2 profile. Each diagram shows its place in the larger model structure by showing the tree structure, and a red circle indicating the diagrams place in the structure.

C.4.1 Abstract Model Views

Figure C.8 is the first model in the abstract section and shows all interfaces being used in the RFP case in an UML2 class diagram. The interfaces are later used to define the roles of the partners in the interaction. The interfaces in this case contains both operations and signal receptors. This is because the implementation should support both synchronous and asynchronous communication. Interfaces are represented with the ball notation.



Abstract
Collaborations
Interfaces
Messages
Process
Roles
Concrete
Adapters
DataTypes
ServiceBindings
Supplier
sendRFP (rfp : RFP)
sendResult (result : HiddenResult)
«signal» RFP (rfp)
«signal» HiddenResult (rfp , isWinner)
ServiceToCustomer
sendRFP (rfp : RFP)
sendResult (winner : Result)
«signal» RFP (rfp)
«signal» Result (winner , proposal)
SupplierRegistry
getSupplierList (category : Category) : SupplierList
ServiceToSupplier
sendProposal (proposal : Proposal)
«signal» Proposal (proposal)
Customer
sendProposalList (list : ProposalList)
sendWinnerDetails (winner : WinnerDetails)
«signal» ProposalList (list)
«signal» WinnerDetails (supplier)

Figure C.8: AuSCL Interfaces.

Figure C.9 defines the messages that are sent between the partners. They are UML2 signals which means that they can be used for asynchronous communication. The diagram is a UML2 class diagram.

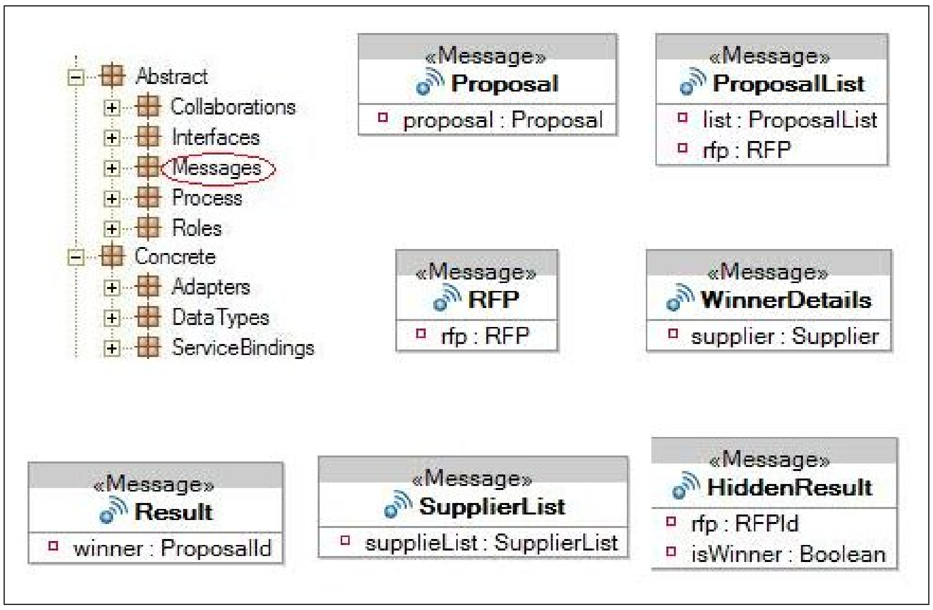


Figure C.9: AuSCL Messages.

Figure C.10 defines the roles in the case. Roles are a combination of provided and required interface. The Customer role provides the Customer interface and requires the ServiceToCustomer interface from another partner.

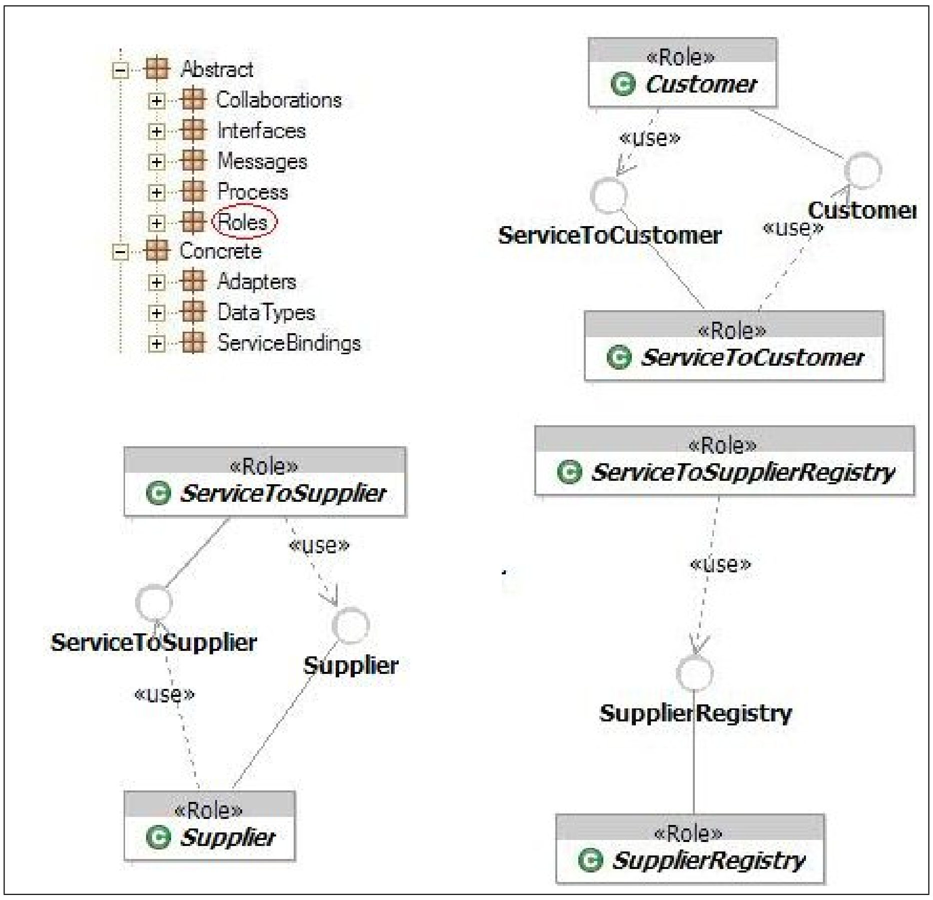
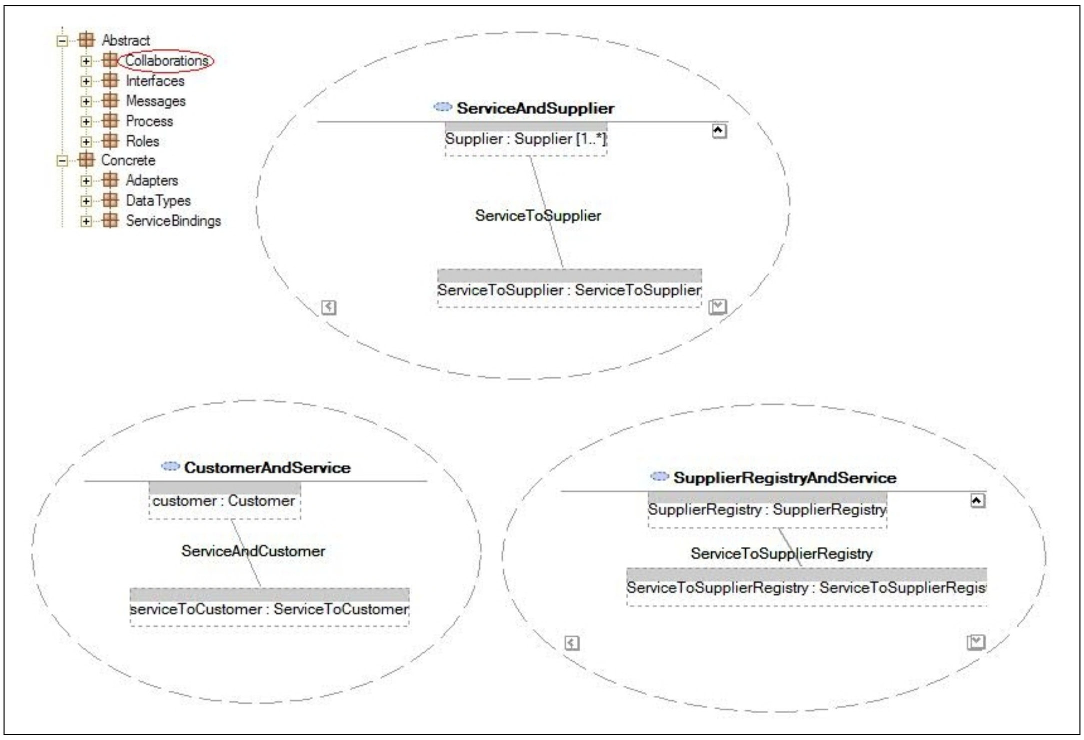


Figure C.10: AuSCL Role Definitions.

Figure C.11 gives an overview of all the collaborations in the service composition. There is one collaboration for each partner-pair, where the collaboration details all interaction that these partners do.

Figures C.12, C.14, C.15 and C.16 specifies the beahviour of the collabora-



Abstract
Collaborations
Interfaces
Messages
Process
Roles
Concrete
Adapters
DataTypes
ServiceBindings
ServiceAndSupplier
Supplier : Supplier [1..*]
ServiceToSupplier
ServiceToSupplier : ServiceToSupplier
CustomerAndService
customer : Customer
ServiceAndCustomer
serviceToCustomer : ServiceToCustomer
SupplierRegistryAndService
SupplierRegistry : SupplierRegistry
ServiceToSupplierRegistry
ServiceToSupplierRegistry : ServiceToSupplierRegis

Figure C.11: AuSCL Collaborations.

tions defined in figure C.11. Sequence diagrams are used to model the message exchanges. Each diagram models the realization of one functional objective. Such as sending rfp's out to all suppliers and getting a proposal in return.

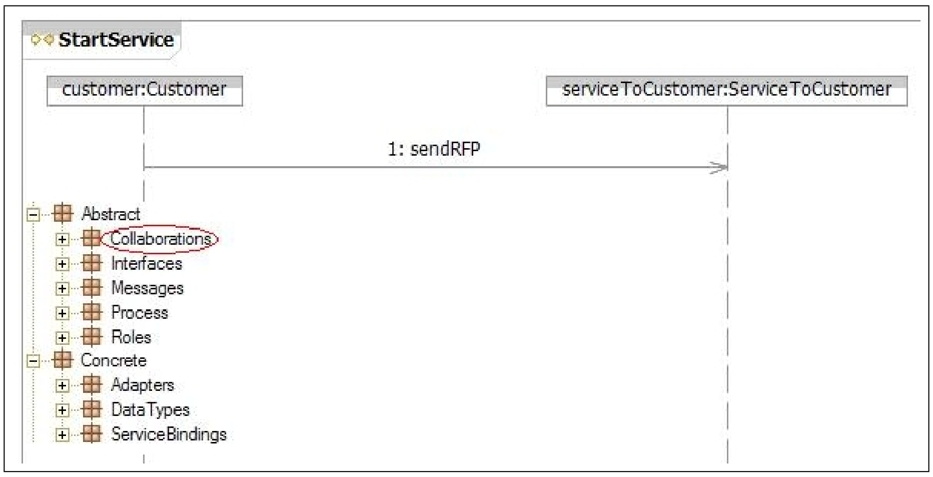


Figure C.12: AuSCL Sequence Diagram StartService Service Interaction. This shows the interaction between the customer and the service , for starting the service.

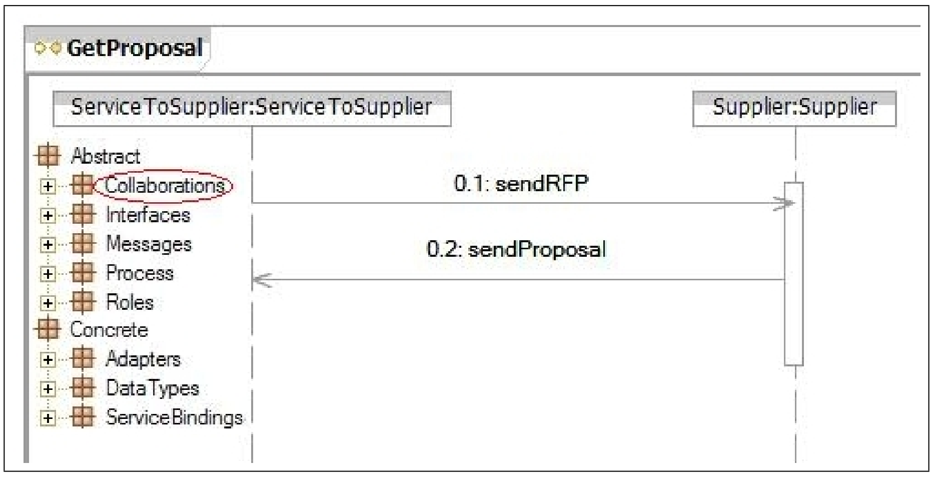


Figure C.13: AuSCL Sequence Diagram Get Proposals Service Interaction. This is the interaction between the service and the supplier for getting the proposals. The service sends a message to the supplier and the supplier sends a proposal back. The communication is asynchronous as the proposal can take relatively long time to create.

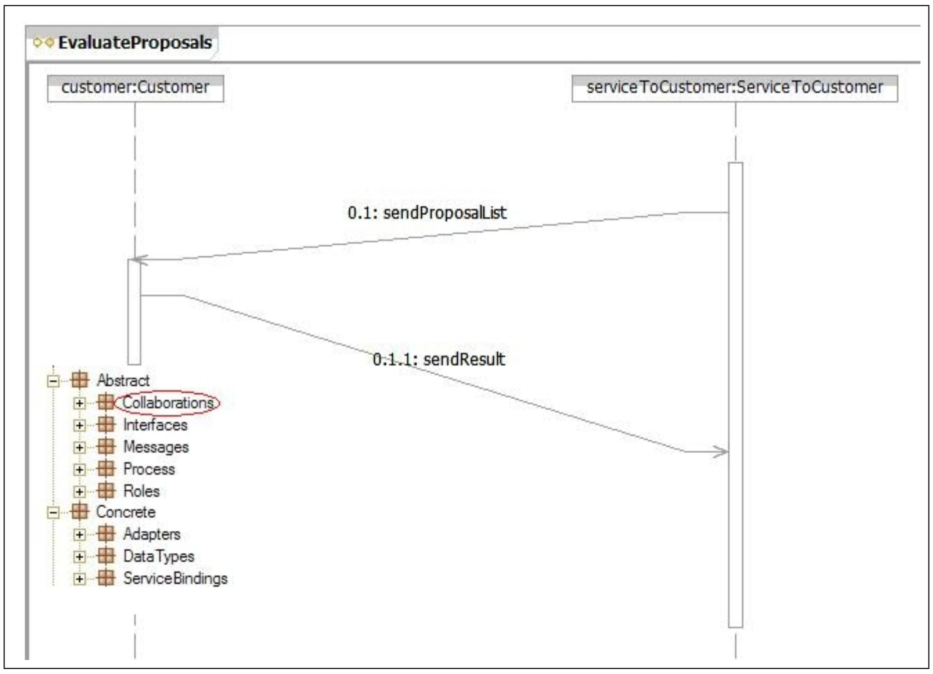


Figure C.14: AuSCL Sequence Diagram Evaluate Proposals Service Interaction. Message interaction between the customer and the service.

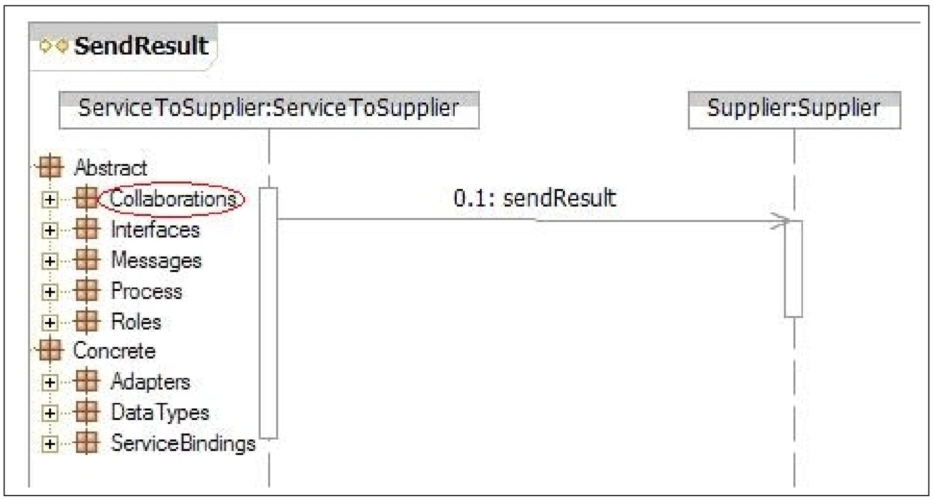


Figure C.15: AuSCL Sequence Diagram. Send Results Service Interaction. The interaction for sening the results back to the suppliers, after the customer have evaluated all proposals.

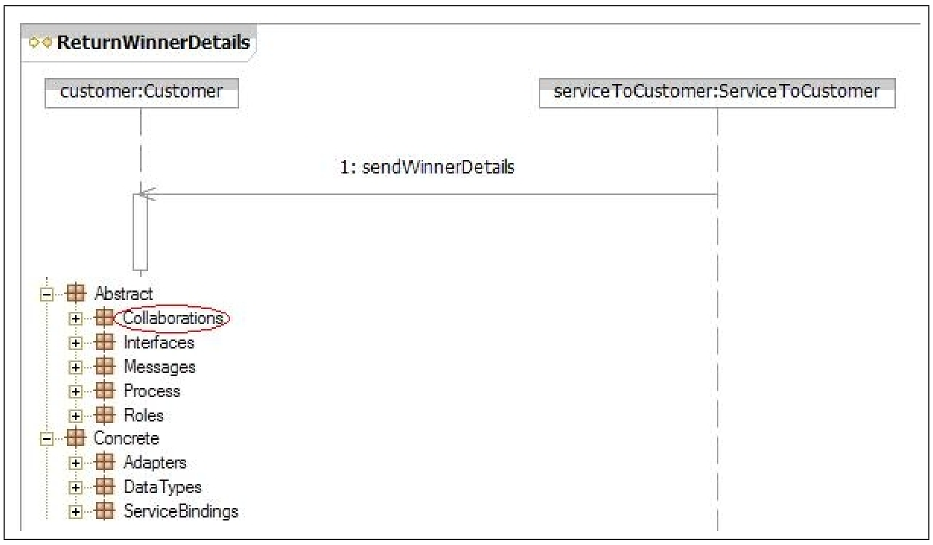
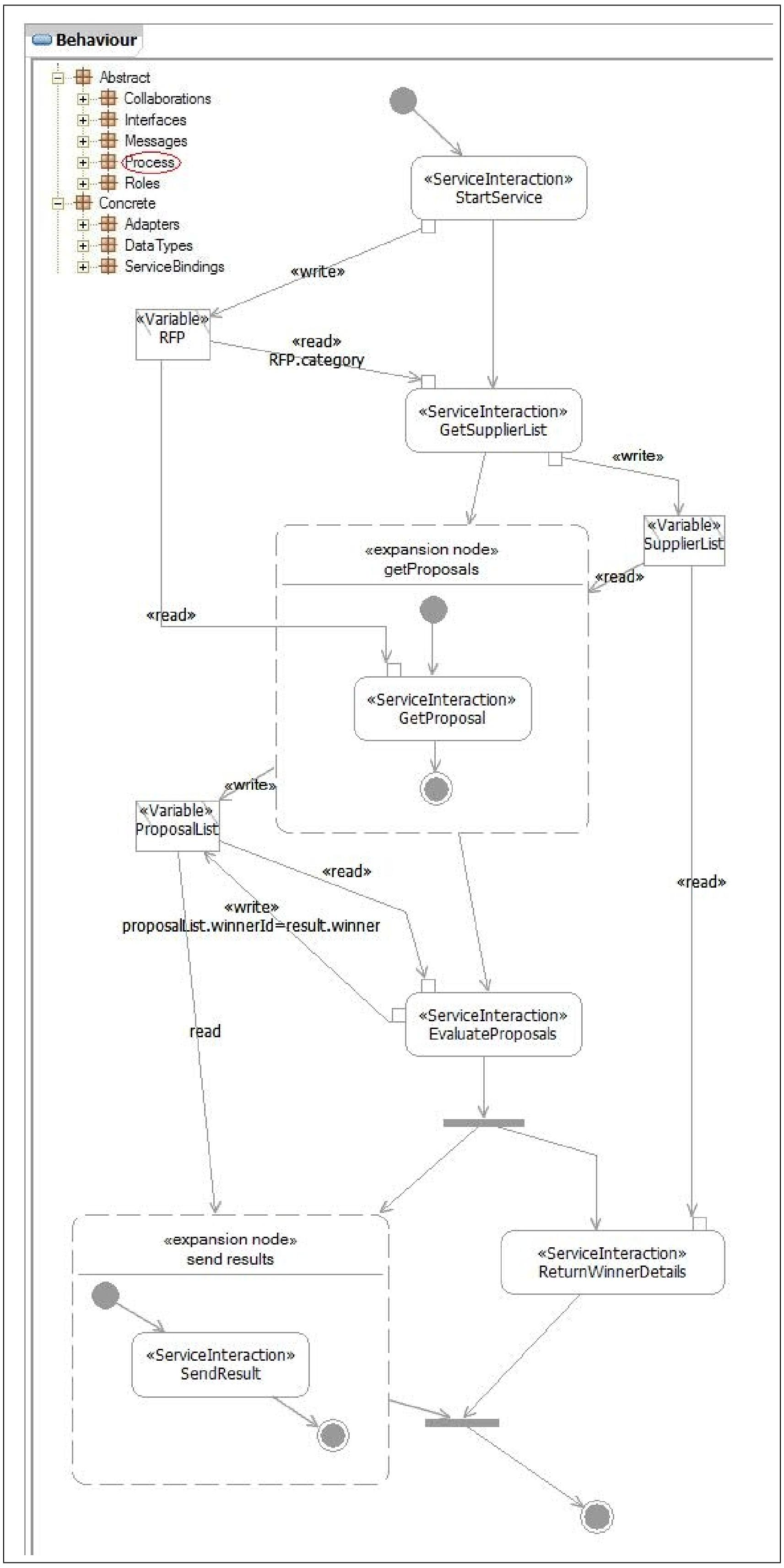


Figure C.16: AuSCL Sequence Diagram Return Winner Details Service Interaction

Figure C.17 is the internal process of the service composition. The process uses several variables, and some complex updating and reading. The getSupplierList task reads the RFP variable, but does only read the category attribute.



Behaviour
Abstract
Collaborations
Interfaces
Messages
Process
Roles
Concrete
Adapters
DataTypes
ServiceBindings
«ServiceInteraction» StartService
«write»
«Variable» RFP
«read» RFP.category
«ServiceInteraction» GetSupplierList
«write»
«Variable» SupplierList
«read»
«expansion node» getProposals
«read»
«ServiceInteraction» GetProposal
«write»
«Variable» ProposalList
«read»
«write» proposalList.winnerId=result.winner
«read»
read
«ServiceInteraction» EvaluateProposals
«expansion node» send results
«ServiceInteraction» SendResult
«ServiceInteraction» ReturnWinnerDetails

Figure C.17: AuSCL Process Diagram.

C.4.2 Concrete Model Views

Figure C.18 is the main model in the concrete part of AuSCL and is the binding between the abstract definition and the concrete service instances. The ports connected to the components specify the roles that the component implements. the components are connected by creating a dependency fro the required interface of one component and the provided interface of another component. As this service composition is of a conversational nature, the components are connected with several interfaces, making it possible for communications to be triggered by both participants.

Figure C.19 defines datatypes used in the service composition. These definitions

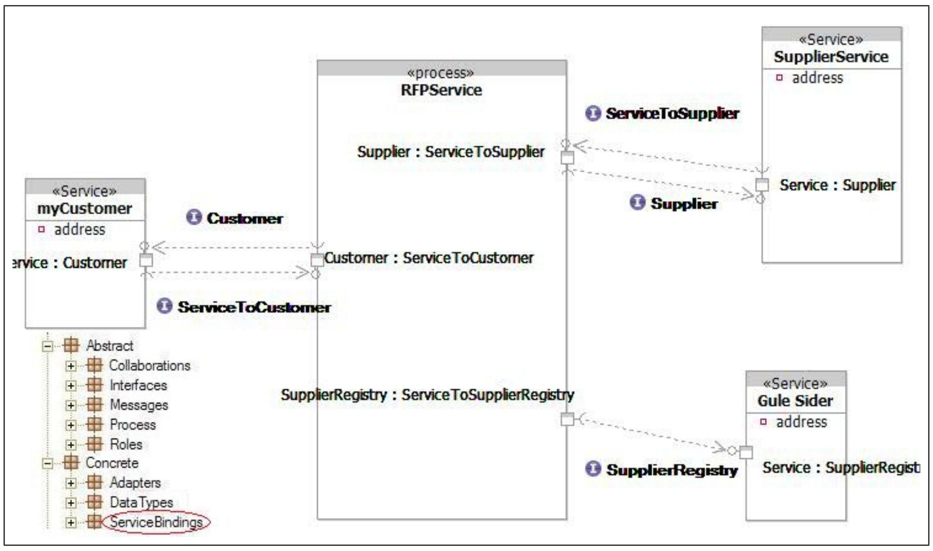


Figure C.18: AuSCL Service Binding Diagram.

are used by the messages defined in the messages model view

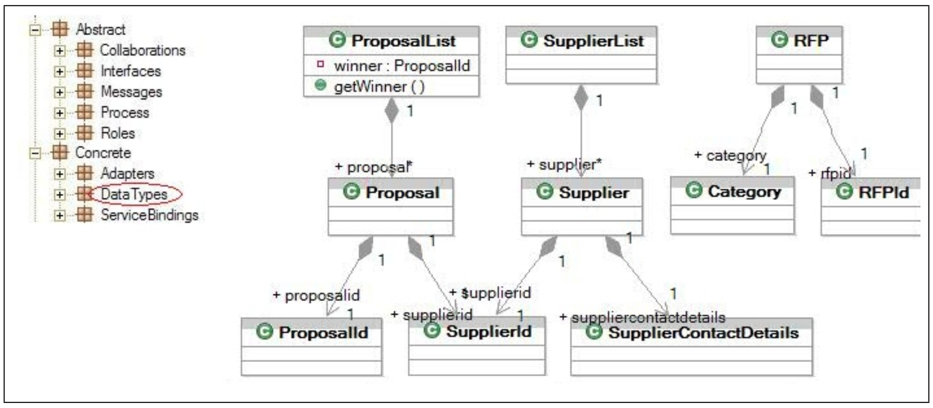


Figure C.19: AuSCL Datatypes Class Diagram.

Figure C.20 shows an example of an adapter between a role definition and a concrete service instance. The name of the operation is different in the two cases and an adapter gives the mapping so that a runtime environment can do the necessary transformation.

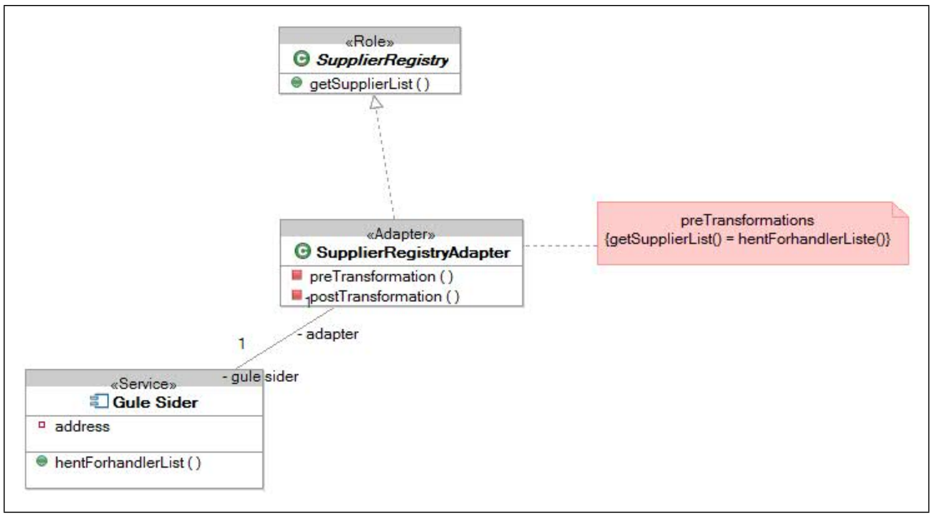


Figure C.20: AuSCL Process Adapter.

Appendix D

Case B - Generic Disk Drive

D.1 Case Description

This case is described one of the subspecifications for the WSRF specification [48] as an example. A disk drive is the stateful resource which can be accessed as a WS-Resource through a web service interface. This WSRF enabled example is different from a pure Web Service example as it contains lifecycle management of stateful resources, in this case by starting and stopping the diskdrive. A use case diagram of the case can be seen in figure D.1

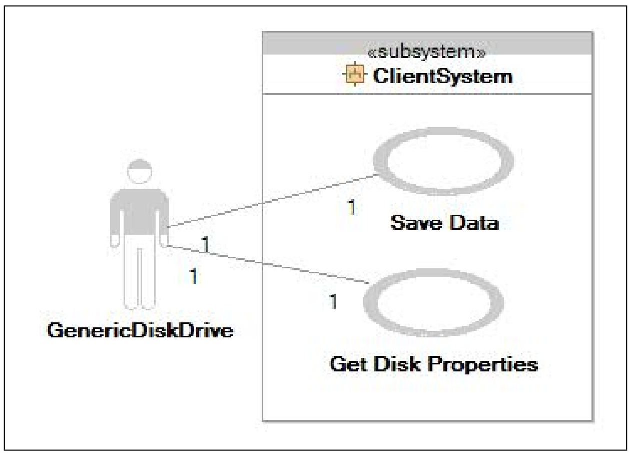


Figure D.1: UML2 Use case description of the Generic disk drive case.

Actors

- DiskDrive - This disk drive contains some resources that describe its state. There are also operations for starting, stopping and storing data on the drive. The drive must be started before data can be stored on it.
- Client - This is the client that wants to store data on the disk.

Behaviour The client starts the disk, gets the properties of the disk to see that these properties indicate the the disk is able to handle storing the relevant file. The the client store the data on the disk, before stopping the disk.

Issues

- Life cycle - The disk drive has a predefined lifecycle, and must be started by an operation before it is used for storing data.

D.2 UML2 Service Composition

This section contains the UML2 models for this case. The UML2 models contains a class diagram specifying the interfaces and classes used in the model, a component diagram used to define the which interfaces are implemented by which components, and a sequence diagram and an activity diagram to model the behaviour.
Figure D.2 shows the interface to an generic disk drive as given in he case description. As this interface is to a stateful resource, the interface contains both attributes and operations. The attributes are the externally visible state of the resource. The ball notation is used for interfaces.
Figure D.3 hows the interface dependencies between the client and the disk

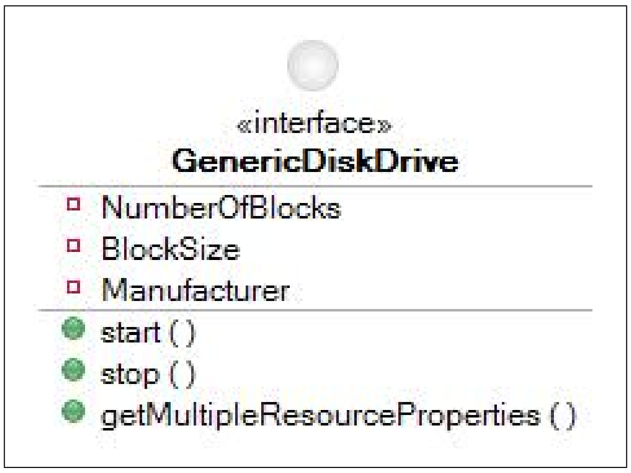


Figure D.2: WSRF Interfaces using UML2 class diagrams.

drive in a UML2 component diagram. The ball/socket notation is used to model provided and required interfaces, with a dependency used to connect a required interface to a provided interface.

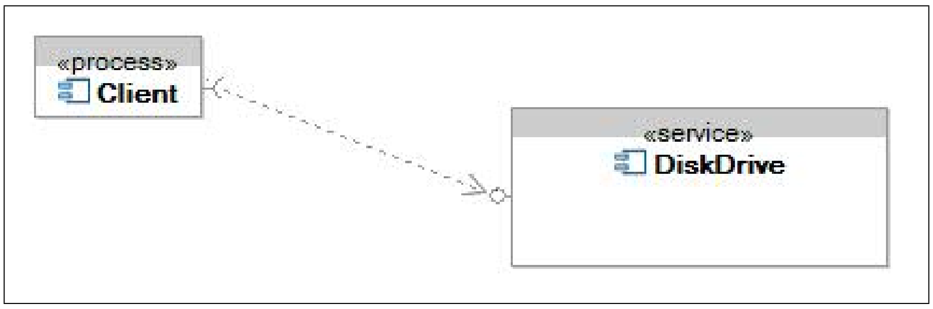


Figure D.3: WSRF Component Dependecies.

Figure D.4 is the protocol of messages going between the client and the diskdrive. One can se here the messages that control the lifecycle of the disk drive with the start message creating a new instance of the diskdrive and the stop message destroying that instance.

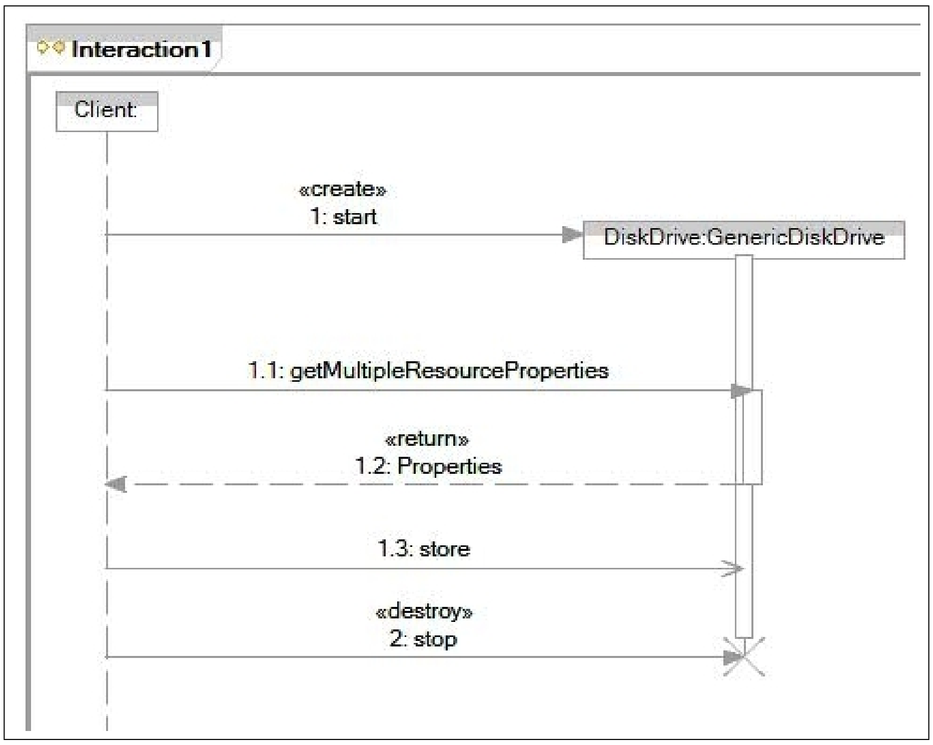


Figure D.4: WSRF Protocol. A UML2 sequence diagram

Figure D.5 is the internal process showing the flow of the service composition, with the «datastore» object holding a copy of the properties that was returned from the disk drive. The actions represent sending and receiving messages with the external partners.

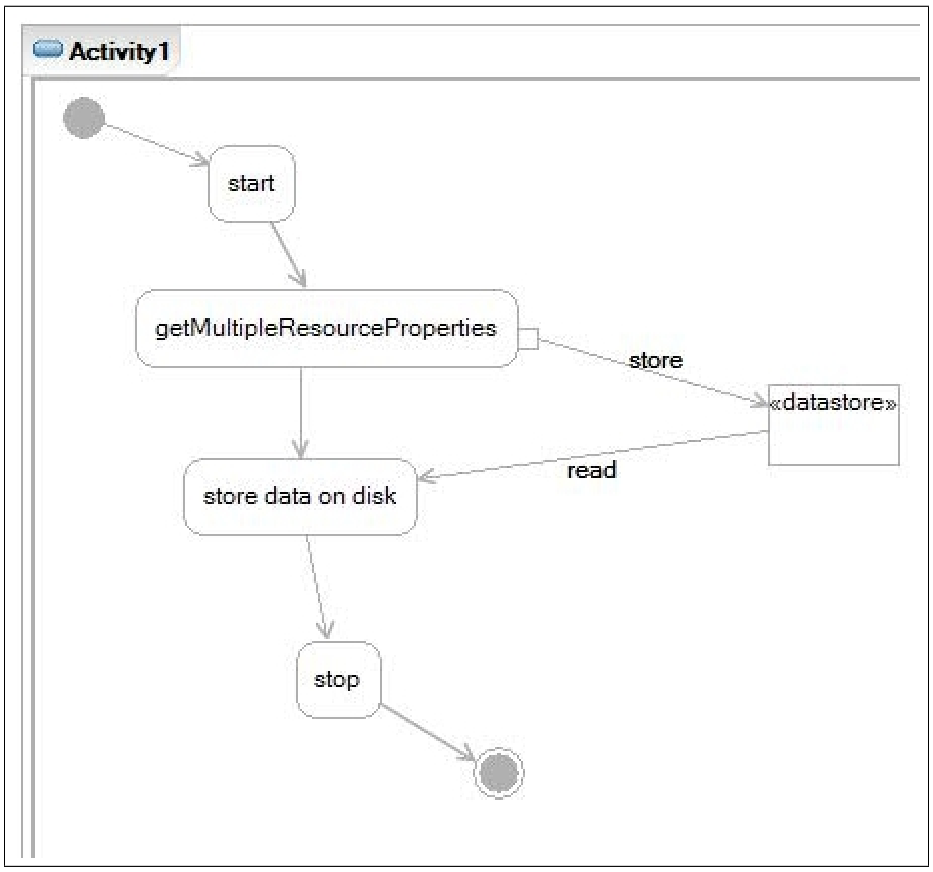


Figure D.5: WSRF Process.

D.3 AuSCL

This section will present a complete AuSCL model as an implementation of the case described in this chapter.

D.3.1 Abstract Model Views

The abstract model views are presented first.
Figure D.6 is the AuSCL interfaces model view. This interface contains both the operations associated with the WS-Resource and its properties. The interface is for a stateful resource.
Figure D.7 defines the roles, and what interfaces each roles depend upon. the

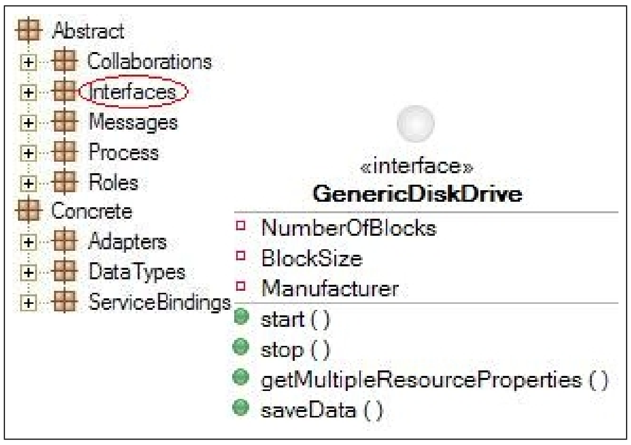


Figure D.6: AuSCL DiskDrive resource Interface.

client role requires the GenericDiskDrive interface, shown in the ball notation, and this interface is provided by the Generic Disk Drive role. The diagram is a UML2 class diagram.

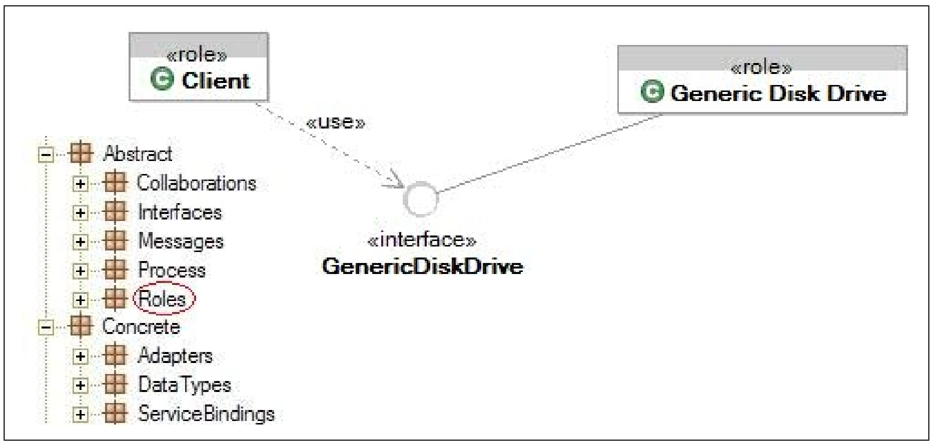


Figure D.7: AuSCL DiskDrive Role Specification.

Figures D.8, D.9, D.10 and D.11 are specifies the interaction between the client and the disk. the model views are separated into units, so that they can be used in the internal process of the service orchestration.

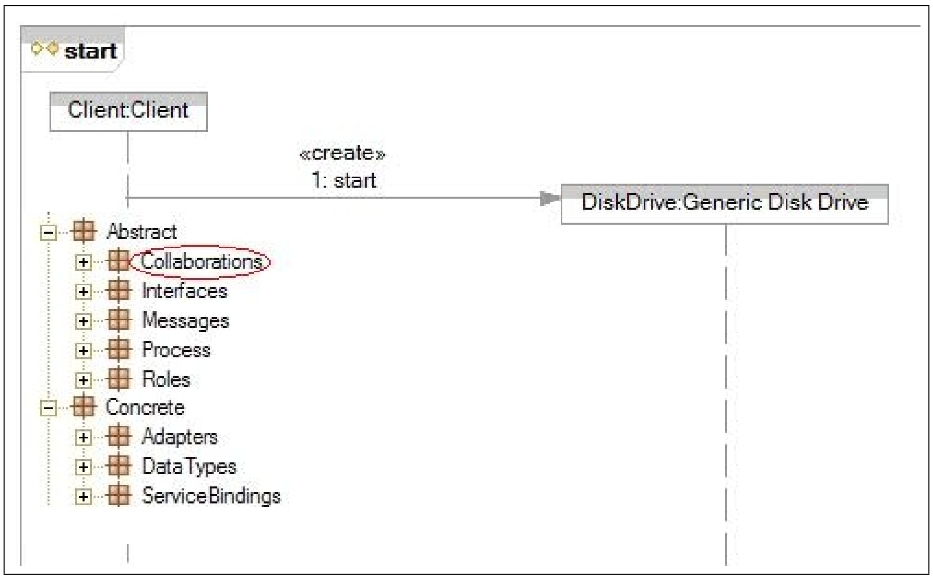


Figure D.8: AuSCL DiskDrive Service Interaction. Message for starting the disk, and creating an instance of the WS-resource.

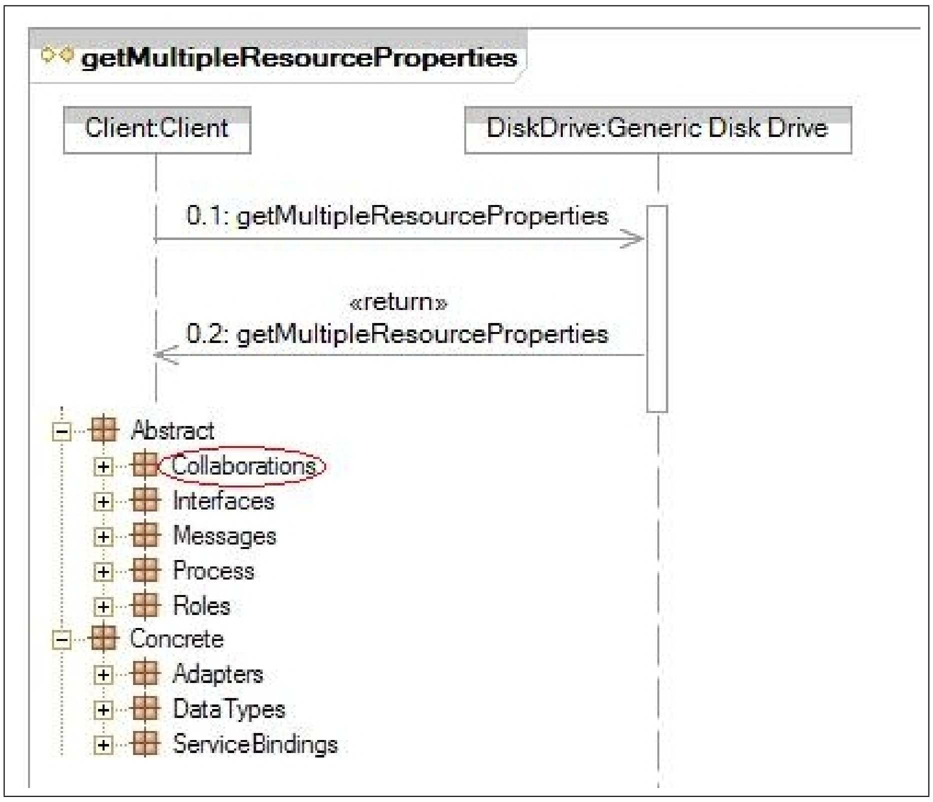

getMultipleResourceProperties
Client:Client
DiskDrive:Generic Disk Drive
0.1: getMultipleResourceProperties
«return»
0.2: getMultipleResourceProperties
Abstract
Collaborations
Interfaces
Messages
Process
Roles
Concrete
Adapters
DataTypes
ServiceBindings

Figure D.9: AuSCL DiskDrive Service Interaction. Serviceinteraction for getting the resource properties for the disk.

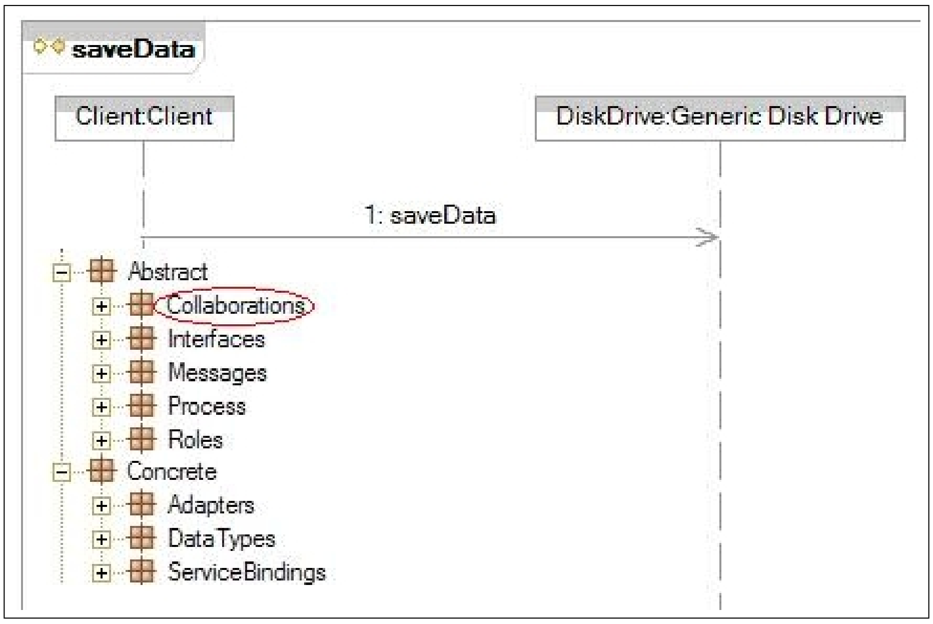


Figure D.10: AuSCL DiskDrive Service Interaction. Interaction or saving data on the disk .

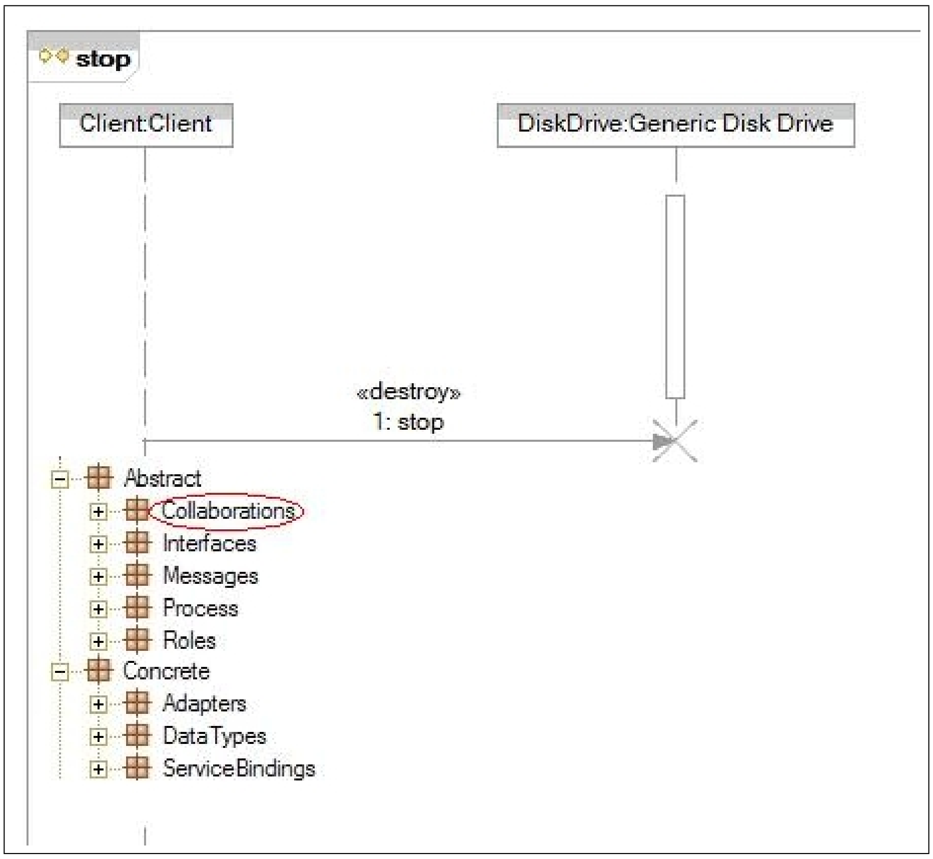


Figure D.11: AuSCL DiskDrive Service Interaction. Stopping the disk and destroying the instance.

Figure D.12 is an UML2 activity diagram. This activity diagrams models the internal process of the service composition. The service interaction actions are defined further in the sequence diagrams seen in figures D.8, D.9, D.10 and D.11.

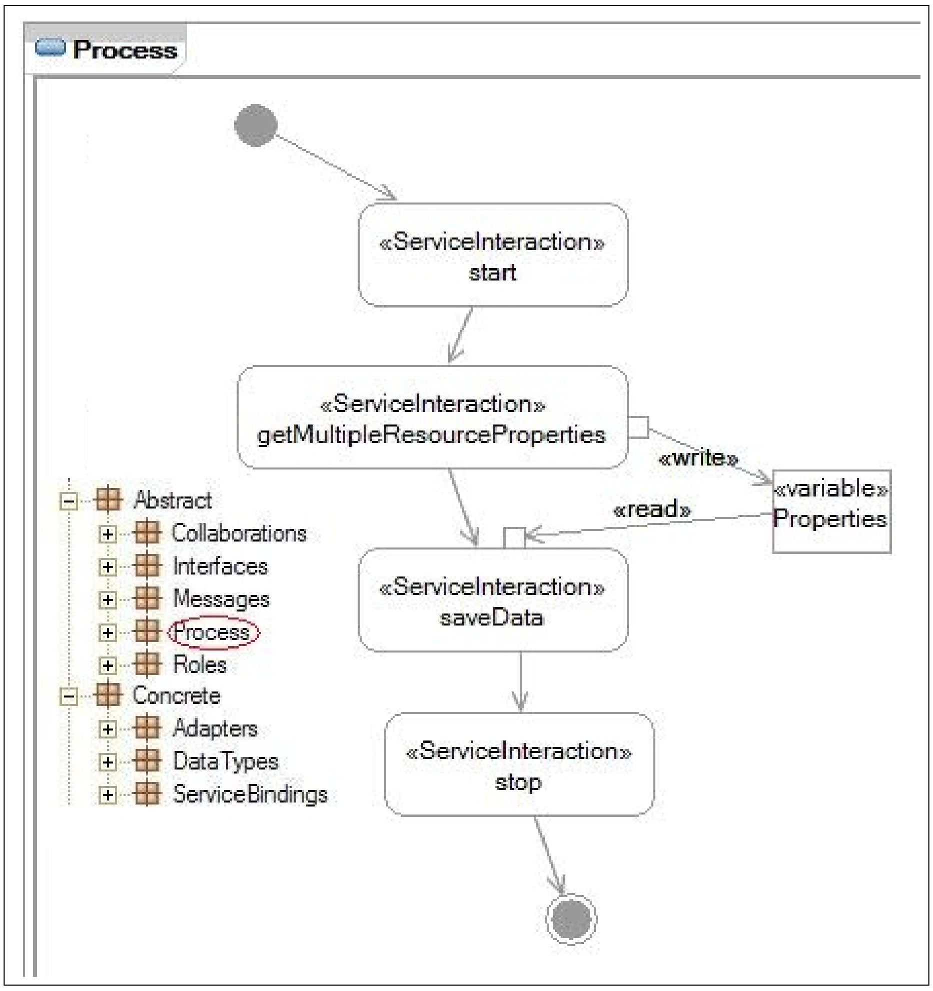


Figure D.12: AuSCL DiskDrive Internal Process.

D.3.2 Concrete Model Views

Figure D.13 binds the abstract roles, shown as ports, to concrete service shown as component. The address attribute of a service point to the implementation of the service, or resource.

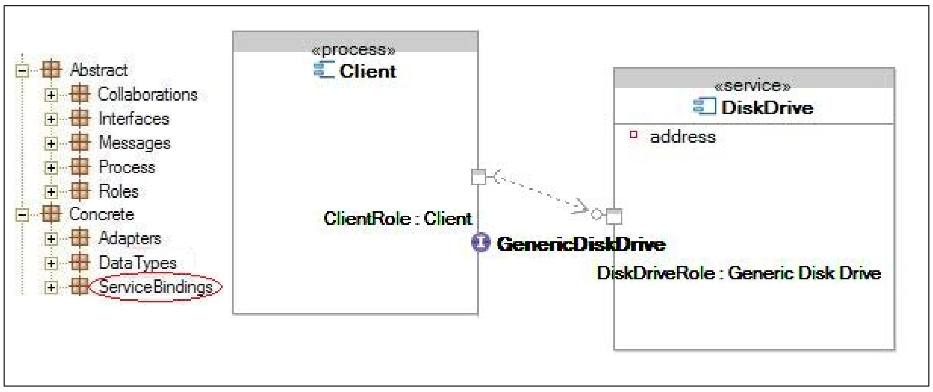


Figure D.13: AuSCL DiskDrive Service Binding.

Appendix E

Case C - Distributed Office Backup

E.1 Case Description

This case is based on a service that backs up a files from a distributed peer-to-peer office collaboration environment to a tape-disk, with some options available for backing up only the newest found version or all found versions of some file. The office environment would be implemented in a P2Protocol such as JXTA and have services for looking up file resources on the network, and the tape-disk would be a stateful resource on a grid, with properties stating the amount of available space on the disk, and operations for starting, stopping and storing. This service composition is made available as a Web Service, so that other applications can use to to get back up of files. A use case diagram of the case can be seen in figure E.1

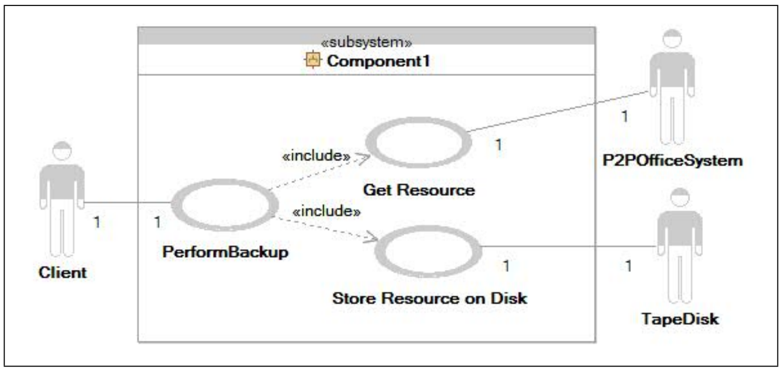


Figure E.1: UML2 Use case description of the Distributed Office Backup.

Actors

- Client - The client want to have backup taken of a file-resource.
- BackupService - The backupservice receives the backup-request and finds the file and stores it on a tape disk. The Backupservice must act as an peer in the P2P network. The service is exposed as a Web Service.

- P2P Office Collaboration Network - This is a P2P based office collaboration application which makes it possible to have a collaborative environment without a central file server. Each peer in the network has the ability to search for file-resources on all the other peers through a search mechanism.
- Tape disk - The tape disk is a backup device. This disk contains properties that describe its state, with information such as available diskspace. The tape disk must be started before properties can be retrieved or data can be stored. Which disk to use should be decided at runtime.

Behaviour The Client sends a backuprequest to the backupservice, containing the name of the file-resource that should be backed up. The BackupService searches for this on the P2P network, and from this gets a list of peers with this resource. If the backup should be of the newest version only, this is downloaded, otherwise alle versions are downloaded. The tape disk is started and the available space is checked against the size of the downloaded resource. If there are enough space available on the disk, the resource is stored.

Issues

- P2P network service invocation. The BackupService must invoke a service on a P2P network.
- Heterogeneous Services - P2P networks, stateful resources on a grid, and Web Services are all used in this case.

E.2 UML2 Service Composition

The UML2 models contains a class diagram specifying the interfaces and classes used in the model, a component diagram used to define the which interfaces are implemented by which components, and a sequence diagram and an activity diagram to model the behaviour.
Figure E.2 defines the interfaces and classes used in the service composition model. The interface for the grid based resource contains both operations and properties.

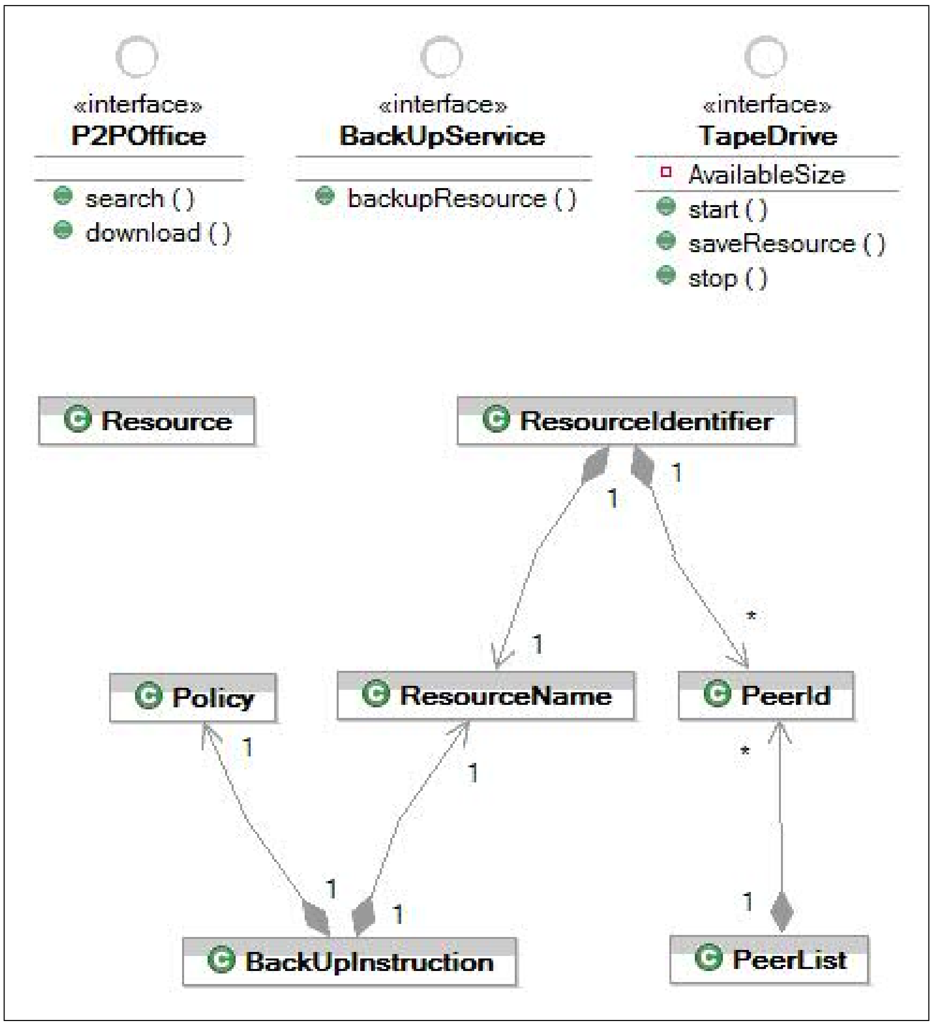


Figure E.2: Interfaces and classes that are used in the distributed office backup case.

Figure E.3 shows the components used in the service composition, and how they provide the interfaces already defined.

Figure E.4 is a UML2 sequence diagrams showing how the partner in the

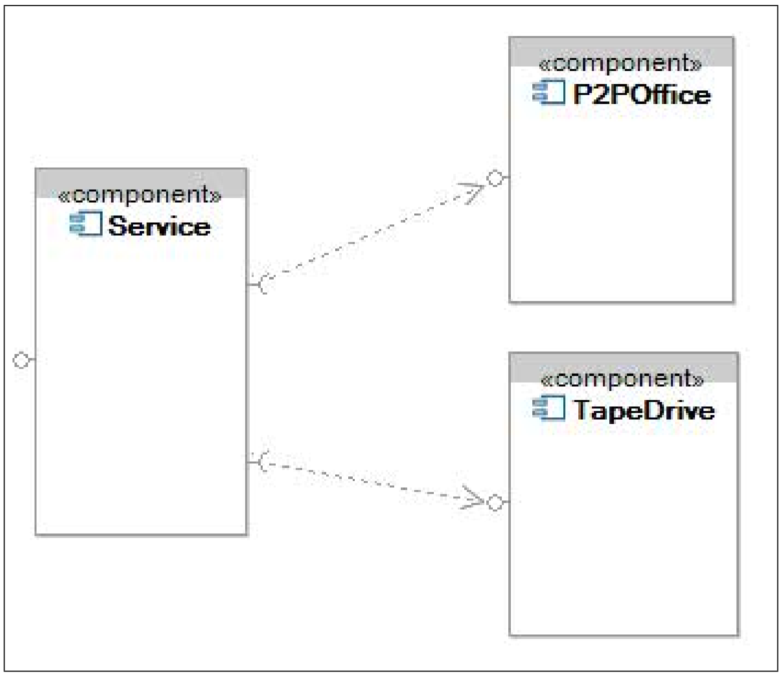


Figure E.3: This diagrams models the components that are used in the distributed office backup case.

service composition interacts. The model view shows the sequence of message interchanges which are expected.

Figure E.5 shows the internal details of the service composition, with tasks for each operation such as sending or receiving messages in a UML2 activity diagram.

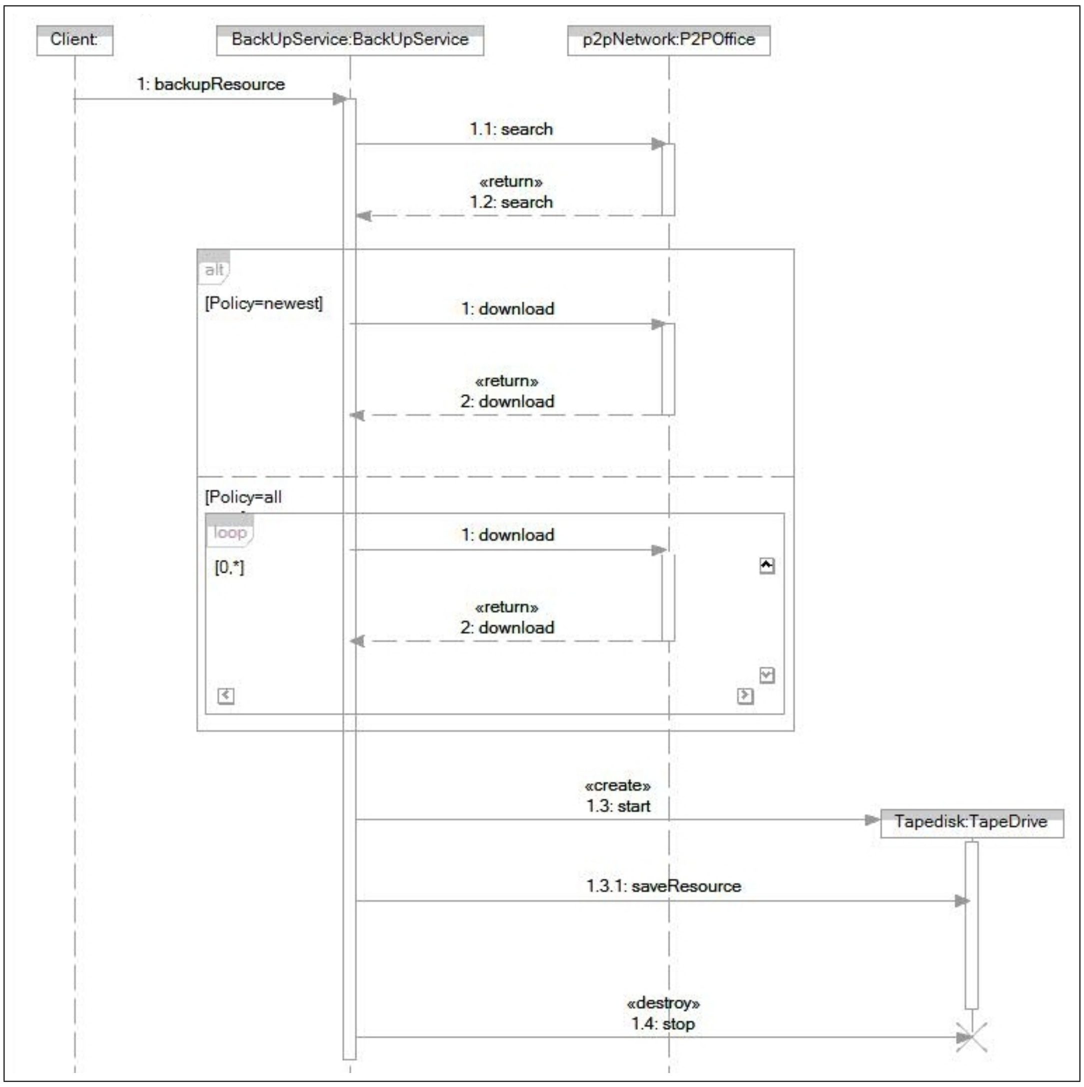


Figure E.4: This sequence diagrams shows the protocol of the complete backup process.

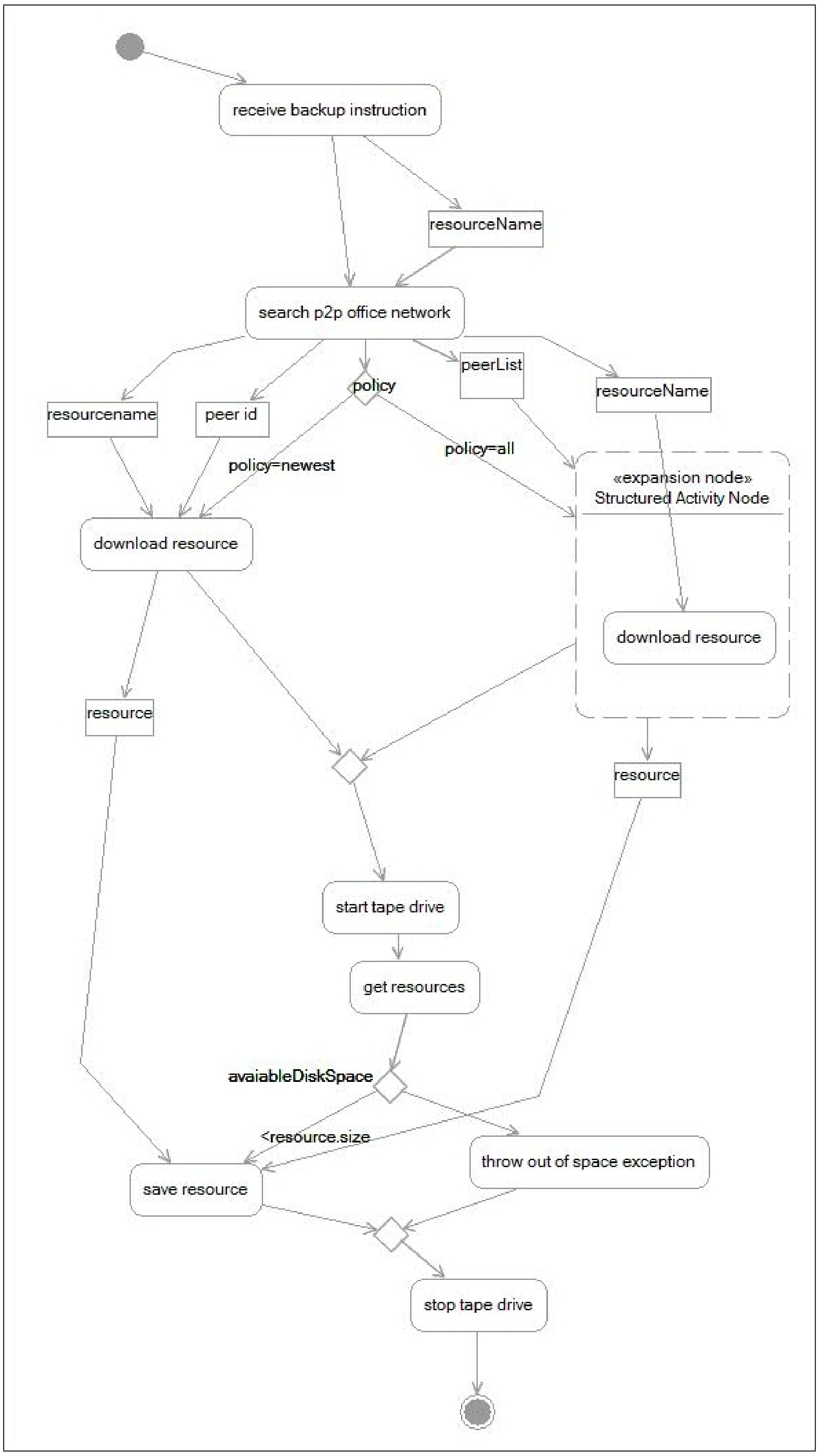


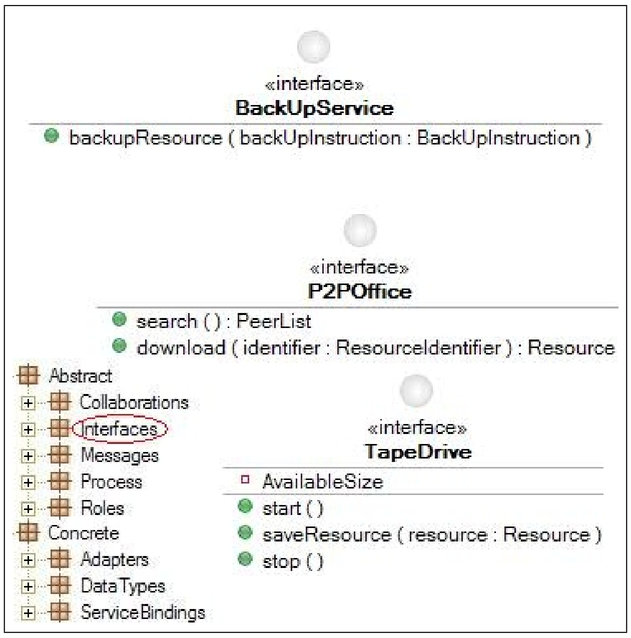
Figure E.5: UML2 activity diagrams showing the internal behaviour of the service composition.

E.3 AuSCL

This section contains the complete AuSCL model implementing the Distributed Office Backup case. The abstract model is presented first.

E.3.1 Abstract Model Views

Figure E.6 defines all interfaces that are used in this case implementation. A standard UML2 class diagram is used. These interfaces will be used in the roles model view, the collaboration model view and the service binding model view.



```
«interface»
BackUpService
backupResource ( backUpInstruction : BackUpInstruction )

«interface»
P2POffice
search ( ) : PeerList
download ( identifier : ResourceIdentifier ) : Resource

«interface»
TapeDrive
AvailableSize
start ( )
saveResource ( resource : Resource )
stop ( )
```

Figure E.6: AuSCL P2POffice Backup Interfaces.

Figure E.7 defines the messages used in this case implementation. The attributes used in the messages are defined in the concrete models, in the data type model view.
Figure E.8 defines the roles, in terms of provided and required interfaces. The

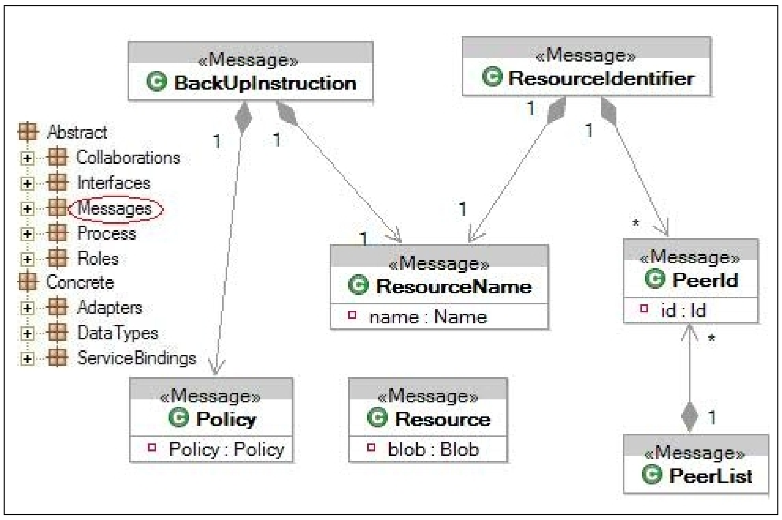


Figure E.7: AuSCL P2POffice Backup Messages.

interfaces are defined separately in figure E.6.

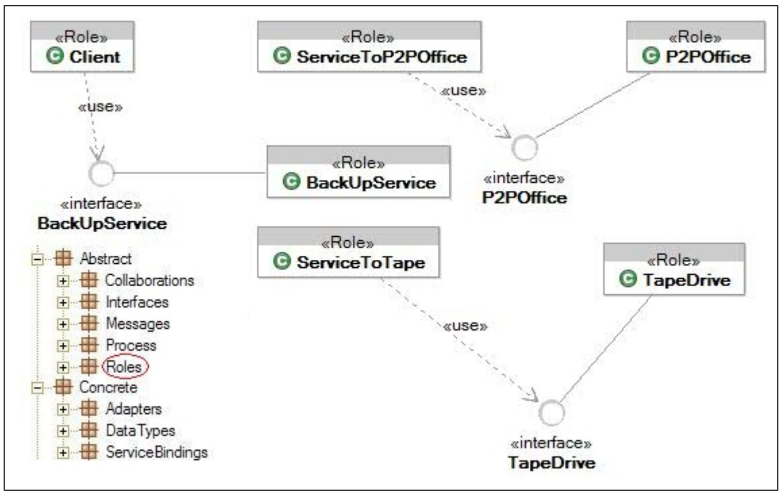


Figure E.8: AuSCL P2POffice Backup Roles.

Figure E.9, E.10, E.11, E.12, E.13 and E.14 specify the behaviour between all the partner in the service composition. These sequence diagrams are used by the process shown in figure E.15.

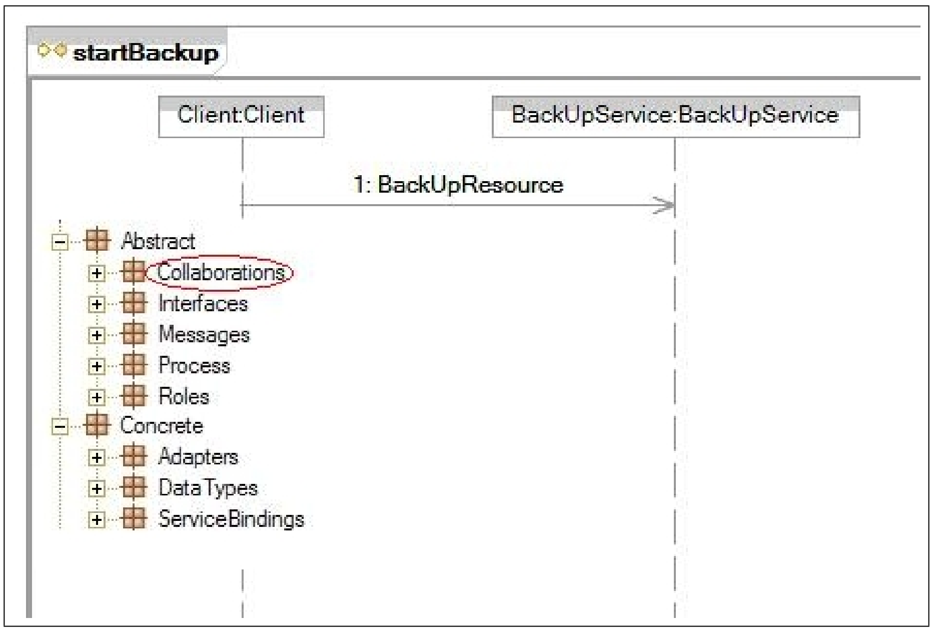


Figure E.9: AuSCL P2POffice Backup Service interaction. The message interactions that are necessary for a client to start the backup service.

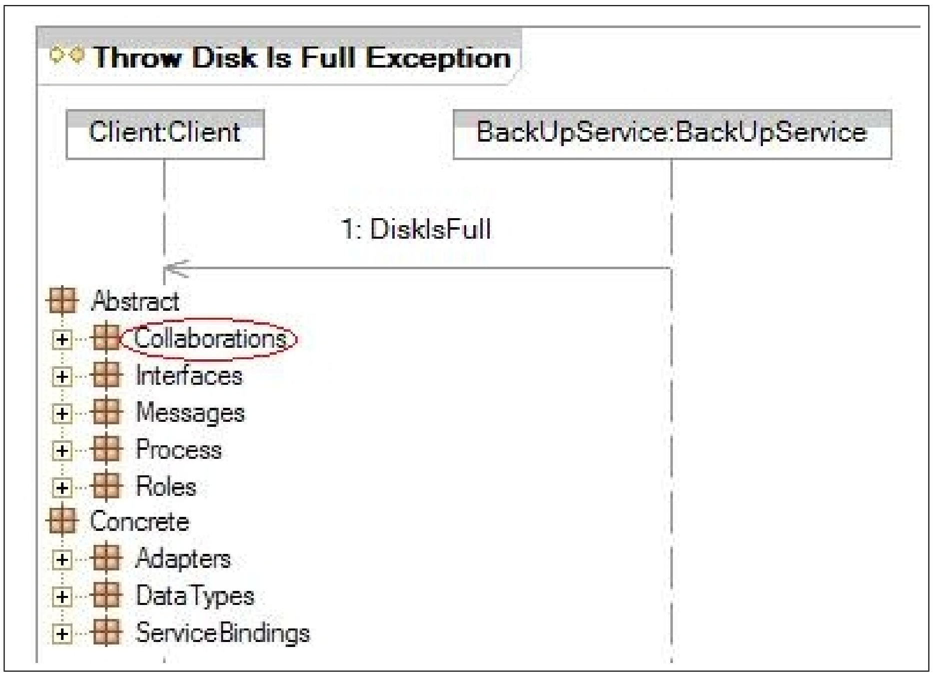


Figure E.10: AuSCL P2POffice Backup Service interaction. A message interaction showing the possibility of receiving an exception if the tapedisk is full.

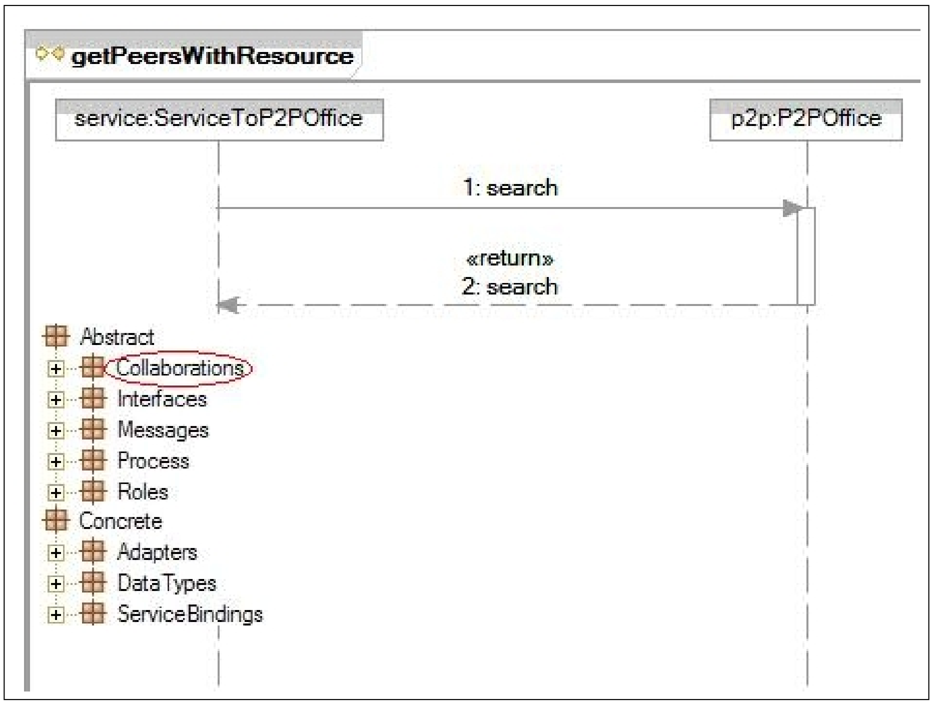


Figure E.11: AuSCL P2POffice Backup Service interaction. Shows the messages involved in searching a p2p network for a specific resource.

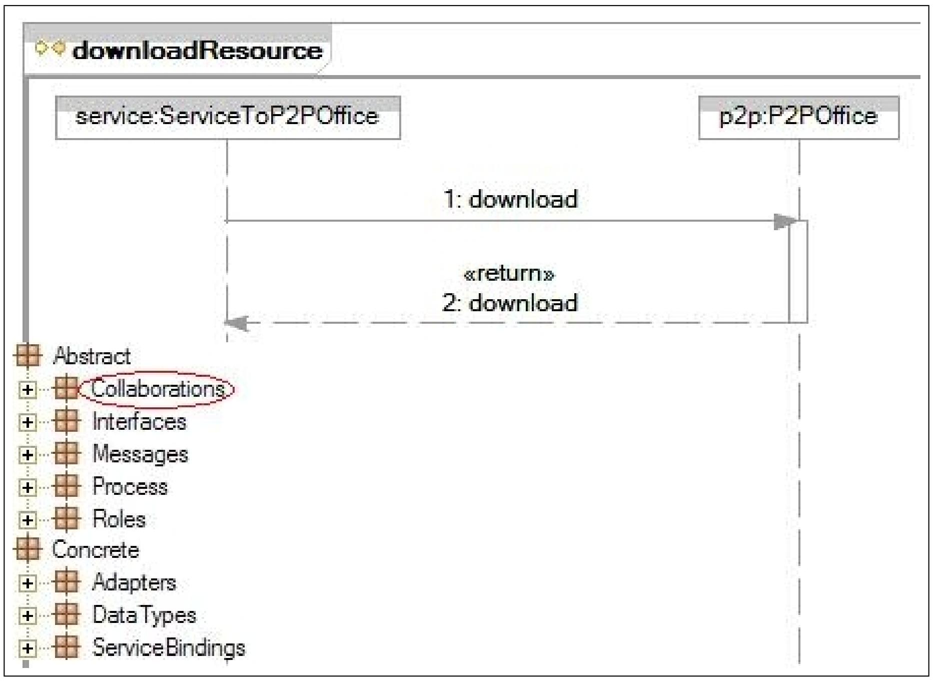


Figure E.12: AuSCL P2POffice Backup Service interaction. Downloading a resource from the p2p network

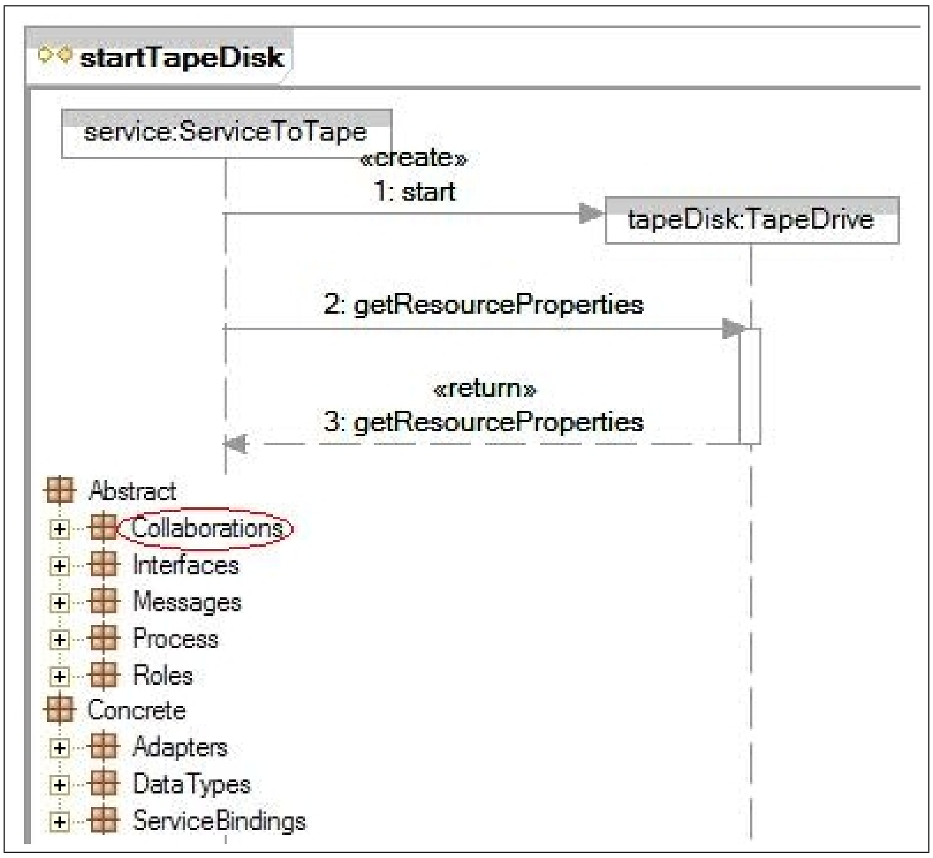


Figure E.13: AuSCL P2POffice Backup Service interaction. This shows the service interaction for starting the tape disk. A create message in sent, and then a request is sent to the disk asking for the resourcepropoerties that the disk have

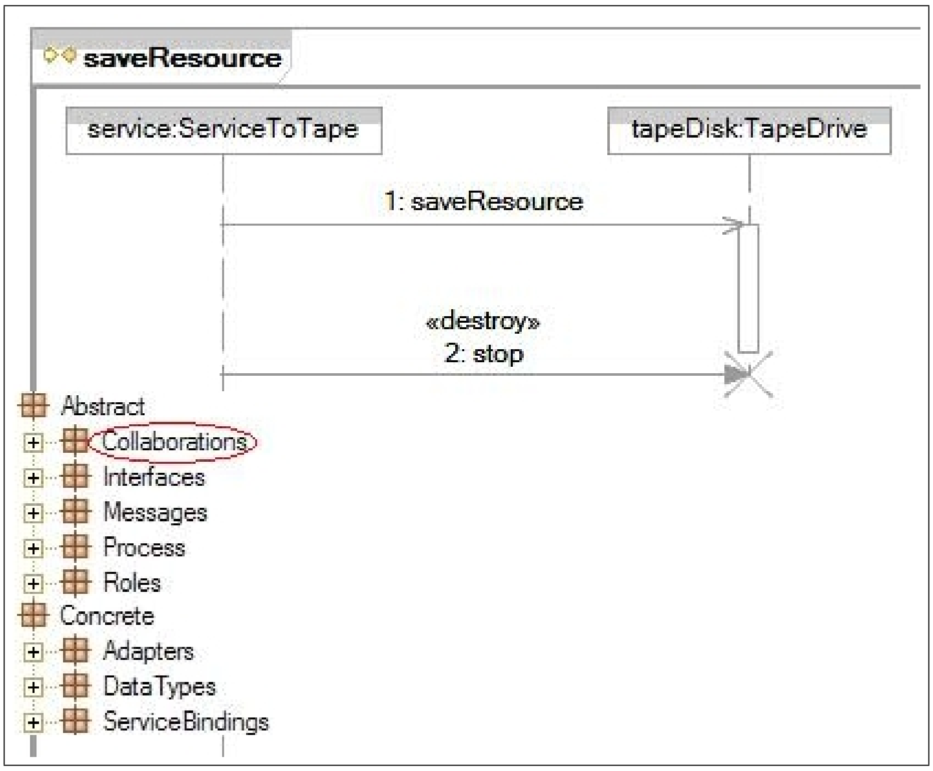


Figure E.14: AuSCL P2POffice Backup Service interaction. This shows the process of saving a resource on the disk and then stopping the disk.

Figure E.15 is the implementation of the flow of getting the resource from the p2p network and then saving the resource to the tape disk. Several process variables are used, and all tasks that involve interactions with another service are stereotypes «ServiceInteraction». An expansion node is used for looping through a collection of dataelements, the peerlist.

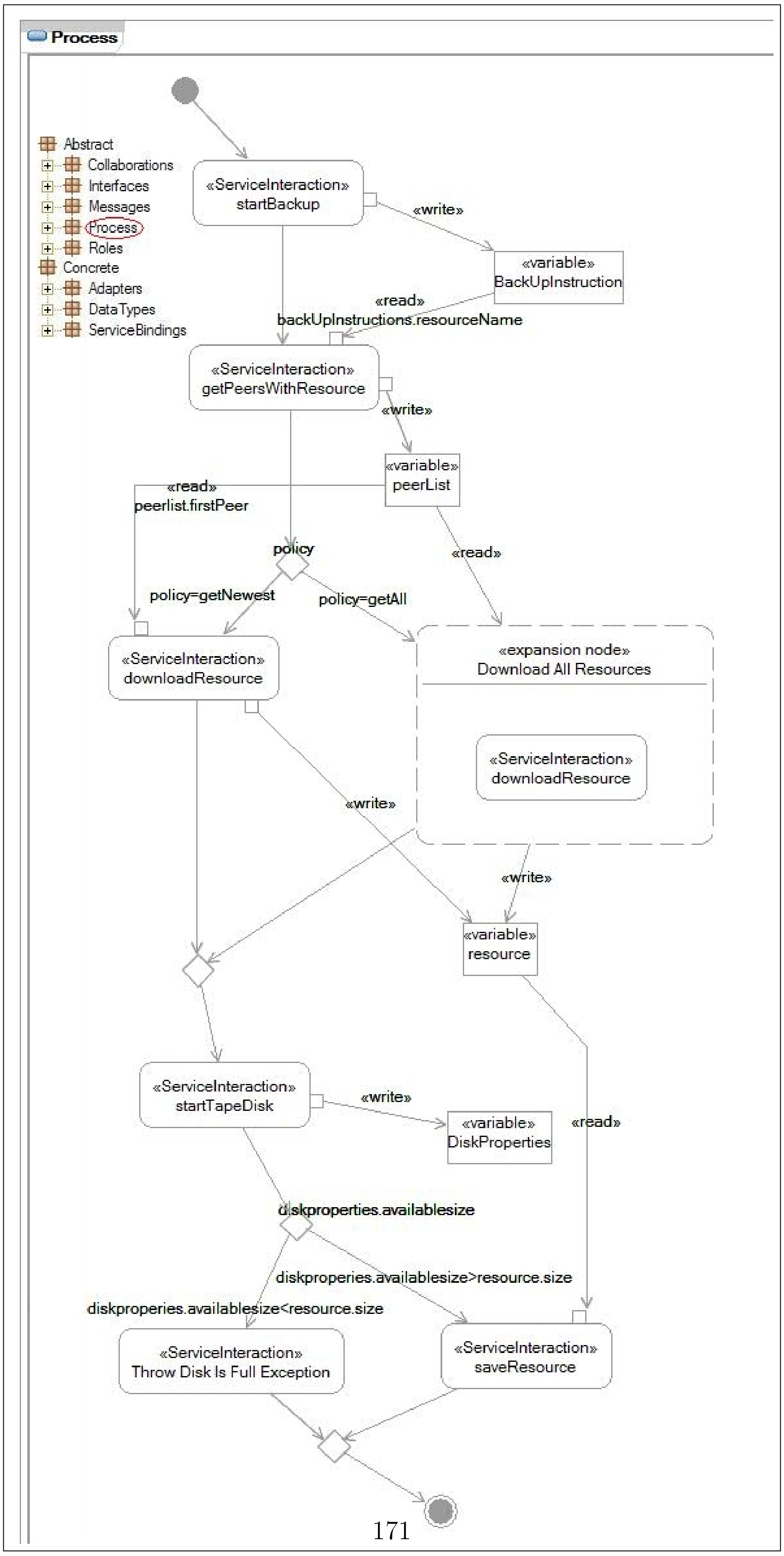

Process
Abstract
Collaborations
Interfaces
Messages
Process
Roles
Concrete
Adapters
DataTypes
ServiceBindings
«ServiceInteraction» startBackup
«write»
«variable» BackUpInstruction
«read» backUpInstructions.resourceName
«ServiceInteraction» getPeersWithResource
«write»
«variable» peerList
«read» peerlist.firstPeer
policy
«read»
policy=getNewest
policy=getAll
«ServiceInteraction» downloadResource
«expansion node» Download All Resources
«ServiceInteraction» downloadResource
«write»
«write»
«variable» resource
«ServiceInteraction» startTapeDisk
«write»
«variable» DiskProperties
«read»
diskproperties.availablesize
diskproperies.availablesize>resource.size
diskproperies.availablesize<resource.size
«ServiceInteraction» Throw Disk Is Full Exception
«ServiceInteraction» saveResource

Figure E.15: AuSCL P2POffice Backup Internal Process.

E.3.2 Concrete Model Views

The concrete model binds the the abstract service composition to service instances. Figure E.16 is the main part of these models, and use component diagrams to bind the roles to concrete service implementations. The tapedisk uses the service query stereotype to indicate that service broker should be used to find tapedisk. One can see the interfaces that are used, and the ports containing the interfaces are the roles defined in the abstract part of the model.

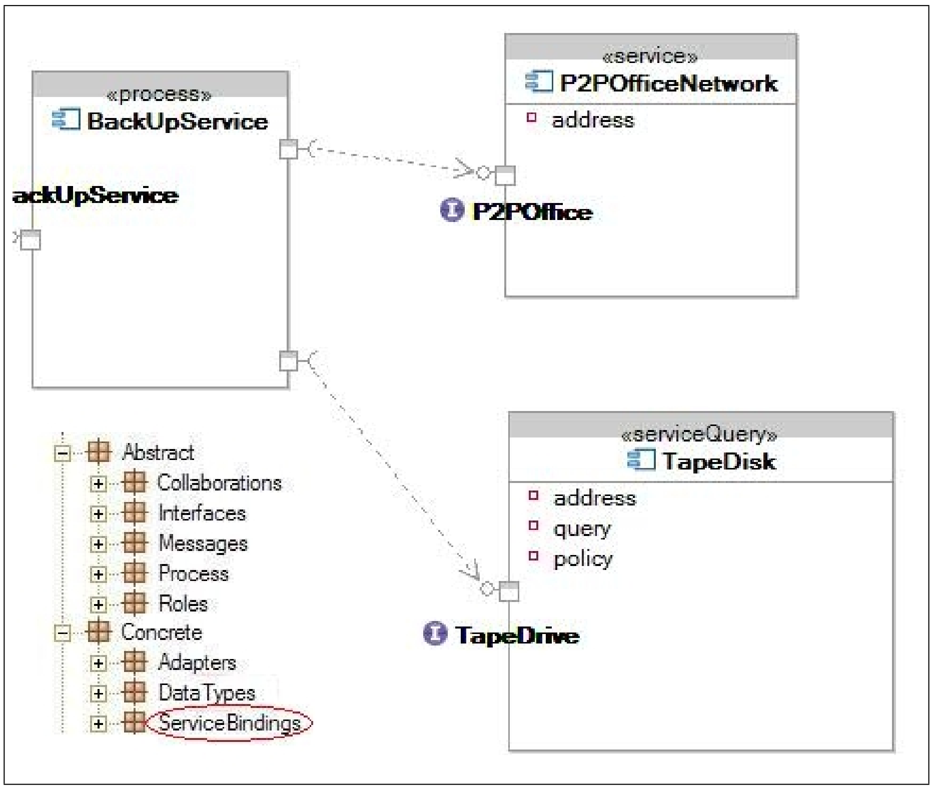


Figure E.16: AuSCL P2POffice Backup ServiceBinding.

Figure E.17 is the concrete specification of the body of the messages defined in figure E.7.

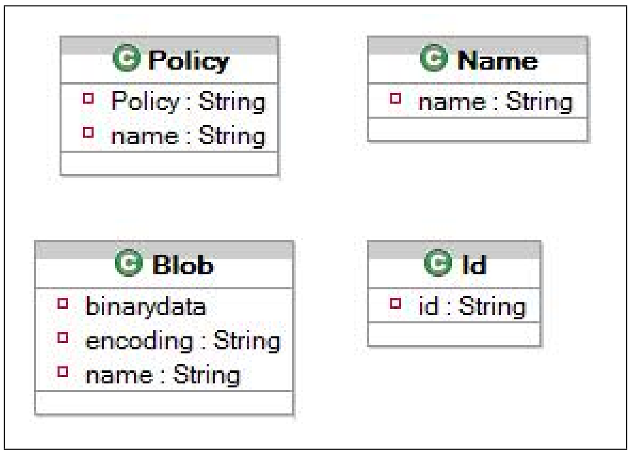


Figure E.17: AuSCL P2POffice Backup DataTypes.