Service and Software Architectures, Infrastructures and Engineering
Small or Medium-scale Focused Research Project
Semantically-enabled Heterogeneous Service Architecture and Platforms Engineering

Acronym
SHAPE

Project No
216408

Deliverable D3.3

D3.3 Metamodel and Language Extension for Semantic Web Services, Agents, P2P and Grid – Initial Version –

Work Package # 3
Leading partner: ESI
Editor: Gorka Benguria
Authors: SINTEF, DFKI, ESI, UIBK
Dissemination level: Public
Date: 14 January 2009
Version: 1.0
## Versioning and contribution history

<table>
<thead>
<tr>
<th>Version</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>TOC</td>
<td>ESI</td>
</tr>
<tr>
<td>0.2</td>
<td>Initial Contents</td>
<td>ESI</td>
</tr>
<tr>
<td>0.3</td>
<td>Agent-based Extensions</td>
<td>Christian Hahn (DFKI)</td>
</tr>
<tr>
<td>0.4</td>
<td>Grid/P2P Extensions</td>
<td>SINTEF</td>
</tr>
<tr>
<td>0.5</td>
<td>Semantic Services</td>
<td>UIBK</td>
</tr>
<tr>
<td>0.6</td>
<td>Peer Review</td>
<td>SINTEF</td>
</tr>
<tr>
<td>1.0</td>
<td>Final version</td>
<td>ESI</td>
</tr>
</tbody>
</table>
Executive Summary

This report presents **Deliverable D3.3 “Metamodel and Language Extension for Semantic Web Services, Agents, P2P and Grid – Initial Version”** of the SHAPE project, including the results of **Tasks T3.5 and T3.6** as defined in the Description of Work.

The objective of this report is to describe extensions to the current version of the core service metamodel introduced in the deliverable 3.1, and revised in the 3.2. The report will focus on the required extensions to the core service metamodel in order to properly support two expected usages of the service modelling. These expected usages are:

- Support metamodel extensions for semantic descriptions. The semantic services are another kind of architecture that should be explore to guarantee that the core service metamodel is also capable of representing semantic service architectures. This task will explore the kind of information that should be added to the core metamodel in order to generate systems that will run over semantic platforms.

- Support metamodel extensions for adaptive systems (multi-agents, P2P and Grid). Apart from the widely adopted web service architecture, there are other service platforms which have proved to be necessary and highly valuable in some cases. These are the service architectures for adaptive systems development. This task will explore the necessary extensions to be performed over the service metamodel in order to support the development of this kind of systems in a seamless way.

The core metamodel is still continuously being updated as result of the continuous feedback received from the implementation and standardisation activities. Therefore, the document will also contain possible updates to the core service metamodel based on its implementation and testing in the user scenarios.

Finally, the document contains the SoaML Profile definition, which is a main SHAPE result in the context of OMG Standardisation activities. SoaML is the new denomination for the UPMS standard in the context of the OMG.

The deliverable D3.3 is composed by this report and by a prototype. The prototype will be the metamodel representation in a formal language such as emof or cmof (Essential MOF or complete MOF).
Table of Contents

VERSIONING AND CONTRIBUTION HISTORY 2
EXECUTIVE SUMMARY 3
TABLE OF CONTENTS 4
FIGURE TABLE 5
1. INTRODUCTION 7
   1.1. DOCUMENT STRUCTURE 7
2. EXTENSIONS FOR SEMANTIC DESCRIPTIONS 8
   2.1. COMPARISON OF SOAML WITH WSMO 8
   2.2. MAPPINGS BETWEEN SOAML AND WSMO – A GLANCE 10
   2.3. PROPOSED EXTENSIONS 11
3. EXTENSIONS FOR ADAPTIVE SYSTEMS 13
   3.1. EXTENSIONS 13
   3.2. EXAMPLE 24
REFERENCES 32
APPENDIX A: SOAML 33
APPENDIX B: SOAML PROFILE 63
# Figure Table

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Extensions covering the deployment of agent instances</td>
<td>14</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Inventoring Service Participant</td>
<td>17</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Supplying Service Realization</td>
<td>17</td>
</tr>
<tr>
<td>Figure 4</td>
<td>P2P Services</td>
<td>18</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Grid Services</td>
<td>20</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Alternative Grid Service model</td>
<td>23</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Configuration with subsets</td>
<td>25</td>
</tr>
<tr>
<td>Figure 8</td>
<td>The Contract Net Protocol using Subset Notation on messages</td>
<td>25</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Abstract Extended Purchase Order Business Process Model</td>
<td>27</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Detailed Extended Process Purchase Order Business Process Model</td>
<td>28</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Business Requirements</td>
<td>29</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Business Interfaces</td>
<td>29</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Service Contracts</td>
<td>30</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Inventoring Service Contract</td>
<td>30</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Supplying Service Contract</td>
<td>31</td>
</tr>
<tr>
<td>Figure 16</td>
<td>GridService contract</td>
<td>31</td>
</tr>
<tr>
<td>Figure 17</td>
<td>SoaML packages</td>
<td>34</td>
</tr>
<tr>
<td>Figure 18</td>
<td>ServiceInterfaces and Participants</td>
<td>34</td>
</tr>
<tr>
<td>Figure 19</td>
<td>ServiceContracts and ServiceArchitectures</td>
<td>35</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Data modeling</td>
<td>35</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Capabilities</td>
<td>35</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Milestones</td>
<td>36</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Agent</td>
<td>36</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Attachment</td>
<td>37</td>
</tr>
<tr>
<td>Figure 25</td>
<td>Capability</td>
<td>38</td>
</tr>
<tr>
<td>Figure 26</td>
<td>Collaboration</td>
<td>39</td>
</tr>
<tr>
<td>Figure 27</td>
<td>CollaborationUse</td>
<td>40</td>
</tr>
<tr>
<td>Figure 28</td>
<td>Port</td>
<td>41</td>
</tr>
<tr>
<td>Figure 29</td>
<td>Diagram 01. OptionalServiceSpecifications</td>
<td>43</td>
</tr>
<tr>
<td>Figure 30</td>
<td>Diagram 02. RoleBindingContract</td>
<td>43</td>
</tr>
<tr>
<td>Figure 31</td>
<td>MessageType</td>
<td>43</td>
</tr>
<tr>
<td>Figure 32</td>
<td>Milestone</td>
<td>45</td>
</tr>
<tr>
<td>Figure 33</td>
<td>Participant</td>
<td>46</td>
</tr>
<tr>
<td>Figure 34</td>
<td>Diagram 01. Simple</td>
<td>47</td>
</tr>
<tr>
<td>Figure 35</td>
<td>Diagram 02. Provider</td>
<td>48</td>
</tr>
<tr>
<td>Figure</td>
<td>Diagram</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>36</td>
<td>03. Consumer</td>
<td>48</td>
</tr>
<tr>
<td>37</td>
<td>04. ContractFulfillServiceArchitecture</td>
<td>48</td>
</tr>
<tr>
<td>38</td>
<td>05. ContractFulFillParticipant</td>
<td>48</td>
</tr>
<tr>
<td>39</td>
<td>06. Behavior</td>
<td>49</td>
</tr>
<tr>
<td>40</td>
<td>07. ComposedParticipant</td>
<td>49</td>
</tr>
<tr>
<td>41</td>
<td>08. SpecificationImplementation</td>
<td>50</td>
</tr>
<tr>
<td>42</td>
<td>09. BehaviorOperation</td>
<td>50</td>
</tr>
<tr>
<td>43</td>
<td>ParticipantArchitecture</td>
<td>50</td>
</tr>
<tr>
<td>44</td>
<td>Property</td>
<td>51</td>
</tr>
<tr>
<td>45</td>
<td>RequestPoint</td>
<td>52</td>
</tr>
<tr>
<td>46</td>
<td>ServicePoint</td>
<td>53</td>
</tr>
<tr>
<td>47</td>
<td>ServiceChannel</td>
<td>54</td>
</tr>
<tr>
<td>48</td>
<td>ServiceContract</td>
<td>55</td>
</tr>
<tr>
<td>49</td>
<td>01. Simple</td>
<td>56</td>
</tr>
<tr>
<td>50</td>
<td>02. Roles</td>
<td>57</td>
</tr>
<tr>
<td>51</td>
<td>03. UseCaseRealzation</td>
<td>57</td>
</tr>
<tr>
<td>52</td>
<td>04. Connectors</td>
<td>57</td>
</tr>
<tr>
<td>53</td>
<td>05. ConstraintsAndObjectives</td>
<td>57</td>
</tr>
<tr>
<td>54</td>
<td>ServiceInterface</td>
<td>58</td>
</tr>
<tr>
<td>55</td>
<td>ServicesArchitecture</td>
<td>59</td>
</tr>
<tr>
<td>56</td>
<td>01. Participants</td>
<td>60</td>
</tr>
<tr>
<td>57</td>
<td>02. ContractsParticipants</td>
<td>60</td>
</tr>
<tr>
<td>58</td>
<td>03. ConnectableElement</td>
<td>61</td>
</tr>
<tr>
<td>59</td>
<td>04. ContractsConnectableElement</td>
<td>61</td>
</tr>
<tr>
<td>60</td>
<td>05. ParticipantsComponent</td>
<td>61</td>
</tr>
<tr>
<td>61</td>
<td>06. ParticipantsCollaboration</td>
<td>62</td>
</tr>
</tbody>
</table>
1. Introduction

The objective of this report is to describe extensions to the current version of the core service metamodel introduced in the deliverable 3.1, and revised in the 3.2. The report will focus on the required extensions to the core service metamodel in order to properly support two expected usages of the service modelling. These expected usages are:

- Support metamodel extensions for semantic descriptions. The semantic services are another kind of architecture that should be explored to guarantee that the core service metamodel is also capable of representing semantic service architectures. This task will explore the kind of information that should be added to the core metamodel in order to generate systems that will run over semantic platforms.
- Support metamodel extensions for adaptive systems (multi-agents, P2P and Grid). Apart from the widely adopted web service architecture, there are other service platforms which have proved to be necessary and highly valuable in some cases. These are the service architectures for adaptive systems development. This task will explore the necessary extensions to be performed over the service metamodel in order to support the development of this kind of systems in a seamless way.

The core metamodel is still continuously being updated as a result of the continuous feedback received from the implementation and standardisation activities. Therefore, the document will also contain possible updates to the core service metamodel based on its implementation and testing in the user scenarios.

Finally, the document contains the SoaML Profile definition, which is a main SHAPE result in the context of OMG Standardisation activities. SoaML is the new denomination for the UPMS standard in the context of the OMG.

The deliverable D3.3 is composed by this report and by a prototype. The prototype will be the metamodel representation in a formal language such as emof or cmof [1] (Essential MOF or complete MOF).

1.1. Document Structure

The submission is organized into the following sections:

- Section 2. Extensions for semantic descriptions: It describes the required extensions to the core service metamodel in order to allow semantic descriptions for services.
- Section 3. Extensions for adaptive systems: It describes the required extensions to the core service metamodel in order to allow the generation of adaptive systems.

Besides, an appendix will be provided with the updated version of the service metamodel, and another appendix will be provided with the updated version of the profile.
2. Extensions for semantic descriptions

2.1. Comparison of SoaML with WSMO

This section provides an overview about the initial thoughts about the extensions in SoaML based on Semantic Web Services, so that it can also represent semantic service architectures. The proposed extensions will allow to generate the Platform Specific Models that will be able to run over semantic platforms.

The SoaML (Service-oriented Architecture Modeling Language) is an OMG specification that has been created in response to UPMS (UML Profile and Metamodel for Services) RFP. The goals of SoaML are to support the service modelling and design in Service-oriented Architectures. The current SoaML specification takes into account the principles of Service-oriented Architectures (SOA) which includes reuse, granularity, modularity, composability, componentization, portability, and interoperability. However, the next steps for the SoaML extensions are to enable it adhering the design principles of Semantic Web Services.

Web Service Modeling Ontology (WSMO) is the conceptual model for realizing Semantic Web Services that provide guidelines that how the semantic descriptions of Web Services should look like. The WSMO follows certain design principles [Roman et. al., 2005] which are briefly described below with a discussion that how they can be taken into account in SoaML.

<table>
<thead>
<tr>
<th>Features</th>
<th>WSMO</th>
<th>SoaML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Compliance</td>
<td>WSMO inherits the concept of URI (Universal Resource Identifier) for</td>
<td>Currently SoaML elements do not contain this information. Therefore,</td>
</tr>
<tr>
<td></td>
<td>unique identification of resources as the essential design principle of</td>
<td>during the transformation process from SoaML to WSML, the user has</td>
</tr>
<tr>
<td></td>
<td>the Word Wide Web.</td>
<td>to specify the URI. If each of the element in SoaML is extended and</td>
</tr>
<tr>
<td></td>
<td>Moreover, WSMO adopts the concept of Namespaces for denoting</td>
<td>provide with a URI property, it would enable the SoaML to become</td>
</tr>
<tr>
<td></td>
<td>consistent information spaces, supports XML and other W3C Web technology recommendations, as well as the decentralization of resources.</td>
<td>Web-compliant.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>On the other hand, the SoaML is based on UML2 profile or instance of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>meta-model, and the concept of namespaces does not exist in it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>However, in SoaML, the attribute is referred as attribute name followed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>by its type (separated with a colon). If a naming scheme based on</td>
</tr>
<tr>
<td></td>
<td></td>
<td>namespaces is introduced in SoaML, it will help in filling in the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>namespace information in WSML (during the process of transformation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>from SoaML to WSML).</td>
</tr>
<tr>
<td>Ontology-Based</td>
<td>Ontologies are used as the data model throughout WSMO to describe</td>
<td>SoaML is supposed to be platform independent, therefore it can not</td>
</tr>
<tr>
<td></td>
<td>resources as well as the data interchanged during service usage.</td>
<td>support any particular language; however, it can support ODM (Ontology Definition Metamodel)</td>
</tr>
<tr>
<td></td>
<td>WSMO itself is categorized as a</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Description versus Implementation</strong></td>
<td>WSMO differentiates between the descriptions of Semantic Web services elements (description) and executable technologies (implementation).</td>
<td>Similar is the case with SoaML that it allows specifying the description independent from implementation, which can be further transformed for Platform-Specific environments to be executed.</td>
</tr>
<tr>
<td><strong>Ontological Role Separation</strong></td>
<td>Users, or more generally clients, exist in specific contexts which will not be the same as for available Web services. For example, a user may wish to book a holiday according to preferences for weather, culture and childcare, whereas Web services will typically cover airline travel and hotel availability. The underlying epistemology of WSMO differentiates between the desires of users or clients and available services.</td>
<td>Similarly, the SoaML allows users to specify their requirements using ‘Agent’ and ‘Participant’ elements and hence gives independence to the service consumers and service provider to model their description independent, i.e. without knowing each other.</td>
</tr>
<tr>
<td><strong>Centrality of Mediation</strong></td>
<td>As a complementary design principle to strict decoupling, mediation addresses the handling of heterogeneities that naturally arise in open environments. Heterogeneity can occur in terms of data, underlying ontology, protocol or process. WSMO recognizes the importance of mediation for the successful deployment of Web services by making mediation a first class component of the framework.</td>
<td>In the same way, in SoaML the notion of mediation is to be introduced once the link between SoaML and ODM (Ontology Definition Metamodel) is defined. The mediation facility will allow defining mappings between different ontology to cover any possible mismatches that may arise.</td>
</tr>
<tr>
<td><strong>Strict Decoupling</strong></td>
<td>WSMO enforces decoupling between WSMO resources that are defined in isolation, meaning that each resource is specified independently without regard to possible usage or interactions with other resources. This complies with the open and distributed nature of the Web. It allows the service consumer and service requestor to specify its requirements and functionality independent of each other.</td>
<td>In the same way, SoaML provides this independence to the user and to the service provide to model their descriptions. For specifying user description, it has the elements like Agent and Participant.</td>
</tr>
<tr>
<td><strong>Ontology</strong></td>
<td>meta-ontology. Therefore, the WSMO based semantic description of Web Services in WSML itself is based on a defined ontology in WSML. WSMO has to support the ontology language defined for the Semantic Web.</td>
<td>[Gašević et. al., 2006] which is also an OMG standard and is language. Using ODM the ontological terms can be represented in SoaML in a language independent way and link it with SoaML.</td>
</tr>
</tbody>
</table>
Execution Semantics

In order to verify the WSMO specification, the formal execution semantics of reference implementations like WSMX as well as other WSMO-enabled systems provide the technical realization of WSMO.

On the other hand, SoaML is also a meta-model that specifies the description of services at an abstract level. At the same time, it should provide certain information (in a platform independent way) that how the SoaML description should be processed. It will require coming up with a UML specification that details the workflow of the execution of SoaML description, in a platform-independent way, which can be used by any particular platform.

Service versus Web service

A Web service is a computational entity which is able (by invocation) to achieve a users goal. A service in contrast is the actual value provided by this invocation. WSMO provides means to describe Web services that provide access (searching, buying, etc.) to services. WSMO is designed as a means to describe the former and not to replace the functionality of the latter.

SoaML also aims to describe the functionality provided by the services, i.e. upon invocation particular functionality can be achieved. However, in SoaML the scope is broader than only focusing Web Services. SoaML is supposed to cover not only Web Services, but also Agents, Grid and P2P services.

2.2. Mappings between SoaML and WSMO – a glance

WSMO has four major top-level elements which are Ontologies, Goals, Web Service descriptions and Mediators [Roman et. al., 2006]. Each of the top-level have been described below:

- **Ontologies** provide the terminology used by other WSMO elements to describe the relevant aspects of the domains of discourse.

- **Web services** describes the computational entity providing access to services that provide some value in a domain. These descriptions comprise the capabilities, interfaces and internal working of the Web service. All these aspects of a Web Service are described using the terminology defined by the ontologies.

- **Goals** represent user desires, for which fulfillment could be sought by executing a Web service. Ontologies can be used for the domain terminology to describe the relevant aspects. Goals model the user view in the Web service usage process and are therefore a separate top-level entity in WSMO.

- **Mediators** describe elements that overcome interoperability problems between different WSMO elements. Mediators are the core concept to resolve incompatibilities on the data, process and protocol level, i.e. in order to resolve mismatches between different used terminologies (data level), in how to communicate between Web services (protocol level) and on the level of combining Web services (and goals) (process level).
There are certain similarities between SoaML and WSMO as both are meta-models for describing services. However, SoaML is generic than that of WSMO. Some of the concepts from SoaML and WSMO are similar to some extent, whereas some are not. However, a mapping from SoaML and WSMO can be meaningful as WSMO is more expressive than SoaML. After analyzing SoaML and WSMO, we have been able to identify some commonalities between the SoaML and WSMO.

For example, Agent element in the SoaML is used to represent user and maintains the life-cycle of the user requirements and preferences. On the other hand, WSMO uses notion of Goal for representation of the user requirements. 

Attachment element in the SoaML is used to represent any attachment that is intended from user to the end-point service. It can be an independent document as a part of message. In case of WSMO, user provides its preferences and message in Goal, therefore information from attachment element from SoaML can be used for end-point service invocation information in WSMO Goal.

In SoaML, Capability element is used for defining behaviour of the system regardless of its implementation. WSMO Web Service description also includes capability element that is used to describe the function of service and hence corresponds with SoaML capability element.

Property element in SoaML is used to distinguish different elements describing service including instances. It can be classified as non-functional properties in WSMO.

Port element in SoaML specifies which particular services from the service providers are to be used. It is like a Web Service offering multiple operations and port element specifies which particular operation of the service to be invoked. In WSMO, Web Service description have two elements, capability and interface, that specify a particular operation of a service.

MessageType in SoaML is used to describe the message itself that is exchanged between the service requestor and service provider. It includes the description of the structure of the message. Ontologies are used in WSMO in order to describe the complex or composite structure of the messages in a meaningful way. Therefore, MessageType element in SoaML corresponds to Ontologies element in WSMO.

RequestPoint element in SoaML describes the use of a service by defining connection point through which a participant makes its request and uses the service. This information is referred as Grounding in WSMO. Web Service description element of WSMO have grounding information that provides the precise address of the end-point service. Hence, RequestPoint element in SoaML corresponds with WS description (Grounding) in WSMO.

ServicePoint element in SoaML describes the connection point through which a user offers its capabilities and provides a service to clients. This information is modelled as capability and interface description as a part of WS description in WSMO.

ServiceContract element in SoaML is the formalization of a binding exchange of information, goods, or obligations between service consumer and service provider. This information is referred as Choreography as a part of WS description in WSMO where it formally specifies the message exchange information with the service.

Based on above mentioned commonalities and comparison between SoaML and WSMO, below we propose the initial directs for extension of SoaML towards Semantic Web Services.

2.3. Proposed Extensions

Based on the comparison of SoaML and WSMO, there have been certain directions identified for the extension of SoaML towards Semantic Web Services.

Ontological support for SoaML:

Relationship between SoaML and ODM (Ontology Definition Metamodel) will be explored as both being the compatible OMG specifications. The connection of ODM with SoaML will allow specification
of ontological elements in SoaML on a platform-independent manner. This information will then be used while generating WSMO description during the model-transformation mechanism. The ODM extension of SoaML will allow specifying the description of certain features of service consumers and service providers, in detailed and precise manner. Although this description will be platform-independent, however, it will be used for describing WSMO description in detail, during the model-transformation process from SoaML to WSMO. The major elements in SoaML that are planned to described using ODM are Capability, Property, RequestPoint, ServicePoint and ServiceContract.

Mediation support for SoaML:

Just like in SoaML, the notion of mediation is to be introduced once the link between SoaML and ODM (Ontology Definition Metamodel) is defined. The mediation facility will allow defining mappings between different ontologies to cover any possible mismatches that may arise. The mapping rules will be specified in a platform-independent way, for the ODM based ontologies, which will be further used for carrying out actual mediation after the WSMO description of services has been generated using model-transformation mechanism.

Web compliance of SoaML:

Currently SoaML elements do not contain this information. Therefore, during the transformation process from SoaML to WSML, the user has to specify the URI. SoaML will be made compliant with the principles of Web by introducing the notion of URI (Uniform Resource Identifier) in order to uniquely identify each and every element in SoaML and also to be able to access and represent the element over the Web, i.e. making it compliant with the principles of Web.

Moreover, SoaML is based on UML2 profile or instance of meta-model, and the concept of namespaces does not exist in it. However, in SoaML, the attribute is referred as attribute name followed by its type (separated with a colon). If a naming scheme based on namespaces is introduced in SoaML, it will help in filling in the namespace information in WSML (during the process of transformation from SoaML to WSML) and hence make it compliant with the principles of Web.

Execution workflow for SoaML:

In order to verify the WSMO specification, the formal execution semantics of reference implementations like WSMX (Web Service Execution Environment) as well as other execution environments for processing WSMO-based descriptions.

On the other hand, SoaML is also a meta-model that specifies the description of services at an abstract level. At the same time, it should provide certain information (in a platform independent way) that how the SoaML description should be processed. It will require an extension in SoaML with a UML-based specification that details the workflow of the execution of SoaML description, in a platform-independent way, which can be used by any particular platform after carrying out platform-specific transformation.
3. Extensions for Adaptive Systems

3.1. Extensions

3.1.1. Agent-based Extensions

Industry is increasingly interested in executing business processes that span multiple applications. This demands high-levels of interoperability and a flexible and adaptive business process management. The general trend in this context is to have systems assembled from a loosely coupled collection of services. These service-oriented architectures (SOAs) appear to be a natural environment in which agent technology can be exploited with significant advantages.

In the remainder of this section, we investigated the similarities of service-oriented architectures (SOAs) and multiagent systems (MASs). For this purpose, we take SoaML and evaluate whether the proposed approach supports modelling of the core building blocks of MASs.

If the evaluation will tell us that SoaML is not sufficient to model the core building blocks of MAS, we will define suitable extensions to make use of more advanced functionalities provided by MASs.

3.1.1.1. Comparison

Instead of comparing each single concept of SoaML and MASs in general, we rather name the core building blocks of MASs and describe in which manner the particular aspect can be expressed using SoaML.

- **Agent aspect** describes single autonomous entities and the capabilities each can possess to solve tasks within an agent system. In SoaML, the metaclass Agent defines a certain type of agents. As the Agent inherits from Participant, an Agent can be considered as entity providing services (i.e. capabilities). This property nicely corresponds to the manner in which agents and capabilities are linked in MASs. Agents in SoaML are specialized because they have their own thread of control or lifecycle. Another way to think of agents is that they are active participants in a SOA system. Participants are UML Components whose capabilities and needs are static. In contrast, Agents should be considered as Participants whose needs and capabilities may change over time. A Participant represents some concrete Component that provides and/or consumes services and is considered as an active class. However, SoaML restricts the Participant’s classifier behaviour to that of a constructor, not something that is intended to be long running, or represent an active lifecycle.

- **Collaboration aspect** describes how single autonomous entities collaborate within MASs and how complex organizational structures can be defined. In SoaML, a Contract indicates roles interacting within this part and how the exchange of messages between these parties is defined, which is mainly done through UML Sequence Diagrams. In SoaML, a ContractFulfillment (UML CollaborationUse) indicates which roles are interacting (i.e., which parts they play) in the contract. The concept of a Contract can be used to model simple collaborations in MASs. However, social units like organizations and groups that are formed by agents during run-time to take advantage of the synergies of its members, resulting in an entity that enables products and processes that are not possible from any single individual are out of scope of SoaML.

- **Role aspect** in MASs covers feasible specializations and how they could be related to each role type. In SoaML, the concept of a role is especially used in the context of Contracts. Like in MASs, the role type indicates which responsibilities the particular entity takes on. However, in MASs, several different notions of the term role can be considered. Often, beside the more domain-related concept of a role, especially in collaborations between agents, social roles are typically used to express the power relationship between participating entities. Furthermore, in MAS, roles are considered as first class entities. Like agents, that can have access to particular capabilities, behaviours, and resources. These more complex characteristics are not part of SoaML, and can even be hardly modelled using pure UML.

- **Interaction aspect** in MASs describes how the interaction between autonomous entities or organizations takes place. Each interaction specification includes both the entities involved and the order which messages are exchanged between them in a protocol-like manner. In SoaML,
contracts or any kind of collaboration are the place where interactions are defined. Like agent interaction protocols, a services contract takes a role centred view of the business requirements which makes it easier to bridge the gap between the process requirements and message exchange. Furthermore, a Contract can have an owned behaviour which may be a UML Sequence Diagram. However, the ability to multicast messages is one feature that is lacking in UML Sequence Diagrams which is an important feature or characteristic of agent interaction protocols.

**Figure 1: Extensions covering the deployment of agent instances**

- **Behavioral aspect** in MASs describes how plans are composed by complex control structures and simple atomic tasks such as sending a message. In SoaML, a ServiceInterface is a UML BehavioredClassifier and can thus contain ownedBehaviors that can be represented by UML 2 Behaviours in the form of an Interaction, Activity, StateMachine, ProtocolStateMachine, or OpaqueBehavior. This nicely corresponds to the manner in which agents’ behaviours—either external or internal—are represented in MASs.

- **Mental aspect** in MASs defines concepts like Beliefs, Intentions, and Goals. These sorts of concepts are not part of the current version of SoaML. However, it might be interesting to support a more representation of the internal of an Agent in SoaML. As mentioned before, the only difference of Participant and Agent in SoaML’s current version is that an Agent has a lifecycle. How the agent’s reasoning is done is not part of the recent specification.

Most of the basic building blocks for MASs are considered in SoaML at least to a degree which makes the modelling of simple “service-oriented” MASs feasible. However, the remainder of this section deals with feasible extensions to provide a more agent-based modelling.

### 3.1.1.2. Proposed Extensions

**Deployment**

An Organization in MAS normally defines the kind of social structure Agents can take part in. It contrast to a Group, the Organization may also act as Agent to the outside. The major advantage of social units like Organizations is that those are formed to take advantage of the synergies of its members, resulting in an entity that enables products and processes that are not possible from any single individual.
To increase flexibility in social structures like Groups or Organizations, Roles are introduced clearly specifying how the set of tasks that need to be performed by the Organizations are distributed among the Agents part of the Organization.

In contrast to most of the object-oriented approaches, the runtime instances are typically dynamically bound to the particular roles which can be in principles be played by the instance’s classifying agent type. The decision which agent instance actually to bind to a role may depend on the agent instances’ availability or other properties.

The extensions to support the deployment of agent instances are depicted in Figure 1.

**Agent Interaction Protocols**

Agent interaction protocols are normally used for the negotiation between agents. While trying to describe complex protocols with UML Sequence Diagrams, we quickly recognized that the ability to multicast messages is the main feature that is lacking.

However, Haugen presented an approach how to solve the multicast of messages problem within sequence diagrams in UML 2, and thus to further improve the FIPA (Foundation of Intelligent Physical Agents) CNP to better express which participants are involved in which sequence of messages. FIPA CNP applies multiplicities on messages. However, Haugen claims that there is not a matter of numbers only, but rather a matter of subsets. To be able to distinguish the participants from each other, and at the same time describe the internal relationship and multiplicity, Haugen introduced configurations with subsets. In fact, this is already available in UML 2, but seldom used in modeling with composite structures. Subsets are constraints on a class property indicating that the defined property is a subset of some other properties defined in a superclass. In order to specify which kinds of participants receive which kinds of message, Haugen introduced subset notation on messages in sequence diagrams. The notation is rather simple. Attached to one (or both) ends of a message, there is a constraint that uses the keyword all followed by a part name. That part must be a subset of the part represented by the lifeline on the message end with the all constraint.

### 3.1.2. Grid and Peer-to-Peer Extensions

The goal of this section is:
- to exemplify the need for the provision of appropriate notations that will cater for the description of Grid and Peer-to-Peer (P2P) services, and
- to specify such notations as part of the UPMS profile.

Below the proposed extensions are summarized:

**Summary of extensions for P2P service modelling**

The set of extensions for the modeling of P2P services includes the notions of:
- **Peer** – it corresponds to the notion of a peer (or a node) in a P2P network which is able to provide and/or consume services
- **PeerGroup** - it corresponds to a logical set of peers in P2P network which all provide a common set of services.

These two concepts have been specified as extensions to the Participant element of the UPMS profile and to the Classifier element of UML respectively.

**Summary of extensions for Grid service modelling**

The set of extensions for the modeling of Grid service modeling include the notions of:
- **GridServiceInterface** – it models the interface of a Grid service which, besides describing the capabilities offered by a service, specifies the types of resources utilized by the service while providing its capabilities.
- **GridService** – it models the connection point through which one may access the capabilities of a Grid service. Yet, for someone to access those capabilities, the required resources should be provided.
- **GridRequisition** – it corresponds to the connection point via which a Grid service requestor utilizes the capabilities of a Grid service and it offers references to the required resources.
• **Resource**—it models the resources utilized by a Grid service.

The first three constructs have been defined as extensions to the **ServiceInterface**, the **Service** and the **Requisition** elements of the UPMS profile respectively.

As regards the specification of the Resource construct, we have followed two approaches:

• the first approach specifies a Resource as an extension of the **UML Class element**,
• the second one specifies a Resource as an extension of the **UML Type element**.

Comments on which of the aforementioned two approaches is more appropriate for the representation of a Resource are most welcomed.

In the context of the UPMS profile, a Resource is represented as an instatiable type consisting of a set of properties which hold its state. A Resource is accessed by a Grid service using the implied resource pattern. The scope of what a Resource can be, is broad and includes either a software or a hardware component.

Although a Resource may possess specific behavior (e.g. in case a Resource is a software component), this is not modeled in the context of WSRF as a feature of the Resource, but rather as a feature of the related Grid service.

In the rest of the document we present:

• an extended version of the Purchase Order Process example that exemplifies the need for the description of P2P and Grid services, where we also utilize the provided constructs for the description of the identified Grid and P2P services;
• the definitions of the constructs used for the description of P2P services, followed by definitions of the constructs used for the description of Grid services based on the first of the ensued approached;
• Finally, we conclude with the definition of the elements used for the specification of Grid services based on the second of the ensued approaches.

### 3.1.2.1. Service Realization

Following the identification of the service interfaces, the next thing to do is the specification of the realizations of those interfaces. Next we identify the providers of the inventorying and the supplying service interfaces.

**Inventory Service**

As it has been specified before, the inventorying interface is realized by a peer-to-peer system. Therefore, the Inventorying service interface is implemented by a Peer such as the one presented in Figure 2 which is part of the WarehouseGroup peer group. The provided requestOrderInventory operation is specified as an opaque behavior of the Inventory Peer.
Supplying Service

The supplying service interface is a GridServiceInterface which apart from the provided capabilities specifies the resources required for their provision. Hence, a participant implementing this interface should provide a reference to the Resource that is needed in order to grasp its functionality.

As it can be seen in Figure 3, the Supplier participant provides a port which implements the Supplying Grid service. The supplying port depends on a ProductionsPlanDB Resource in order to provide its capabilities. The requestSupplyPlan operation of the Supplying Grid service is specified as an opaque behavior of the Supplier participant.
3.1.2.2. P2P Service Class Description

![Figure 4: P2P Services](image)

**Peer**

A Peer is a node of a peer-to-peer network that acts both as a provider and requester of services offered by the nodes of the network. Usually, all peers in a network provide symmetric functionality and are organized according to specific network patterns e.g. hierarchical, mesh, etc.

**Generalization**

- Participant

**Description**

Peers are autonomous, independent software systems that communicate and collaborate with each other over the network. They share resources such as files, computation power or storage space with each other, thus facilitating the utilization of resources which reside at the edges of the network.

Peers may participate in more than one peer group which are logical sets of peers in a peer-to-peer network.

**Attributes**

No additional attributes.

**Associations**

- `peergroups::PeerGroup[*]` Indicates the set of peer groups where a peer participates in.

**Constraints**

A service provided by a Peer can be consumed by another Peer only.

**Semantics**

A Peer provides for the description of a node of a Peer-to-Peer network. Peers are autonomous, independent software systems that communicate over the network for sharing resources such as
computation power, files, storage space, etc. Peers provide services that can be consumed by other peers only and are normally organized according to specific network patterns such as mesh, hierarchical pattern, etc.

A Peer in UPMS extends the Participant which represents some concrete Component providing and/or consuming services. A Peer constrains the ability of a Participant to provide services that can be consumed by anybody, by dictating that services provided by Peers can only be consumed by other peers in the same network.

**Notation**

A Peer can be designated using the Component or Class/Classifier notation including the «peer» keyword.

**Additions to UML 2.X**

Peer is a new metaclass in UPMS extending UML2 Component with new capabilities.

**PeerGroup**

A PeerGroup is a logical group of peers, which may all provide a common set of services.

**Generalization**

- Classifier

**Description**

A PeerGroup is a logical set of peers. They may act as boundaries for various aspects such as security, or the utilization and provision of specific functionality.

**Attributes**

No additional attributes.

**Associations**

- `providedServices::Service[*]` Indicates the set of services that the containing peers should provide.
- `peers::Peer[*]` Indicates the set of peers which are members of a peergroup

**Constraints**

No additional constraints

**Semantics**

A PeerGroup is a logical aggregation of a set of Peers which accommodate common features, e.g. by providing a common set of services. Such services are normally called Group Services since they are dictated by the group.

A PeerGroup may also be regarded as the boundaries of a set of peers for various aspects such as security, or the provision of specific functionality.

**Notation**

TBS

**Additions to UML 2.X**

PeerGroup is a new metaclass in UPMS extending UML2 Classification.
3.1.2.3. Grid Service Class Description

**GridRequisition**
A point through which a consumer accesses capabilities provided by a Grid service provider.

**Generalization**
- Requisition

**Description**
A GridRequisition is a mechanism through which a GridService consumer accesses the capabilities of a Grid service and provides the Resources needed by a Grid service to perform its functionality. A GridRequisition specifies the required capabilities and the Resources it provides via the GridServiceInterfaces which defines its type.

Similar to the Requisition, a GridRequisition can be considered as the point of interaction between a Grid service consumer and a Grid service provider. A GridRequisition extends the Requisition with the ability to consume GridServices whose type is dictated by the associated GridServiceInterfaces and by specifying the Resources that these Grid services require for their operation.

**Attributes**
No attributes

**Associations**
- protocol:GridServiceInterface [*] Specifies the GridServiceInterfaces which define the capabilities that the GridRequisition is expected to consume and the Resources that it should provide in doing so
- /specifiedResources:Resource [*] Identifies the Resources that the GridRequisition will reference to a Grid service in order to utilize its capabilities. These Resources are derived by the Resources
that the associated GridServiceInterfaces dictate as required when utilizing their specified functionality.

**Constraints**
No Constraints

**Semantics**
The interfaces required by a GridRequisition are specified by the associated GridServiceInterfaces. The Resources referenced by a GridRequisition are those that are required by the associated GridServiceInterfaces.

**Notation**
TBS

**GridService**
A GridService may be defined as a loosely coupled Service with a Resource.

**Generalization**
- Service

**Description**
A Grid service (WS-Resource in the WSRF specification) is defined as a loosely coupled service with a stateful resource 0. This coupling is expressed as an association of an XML document that has a defined type, with a service interface and is addressed and accessed according to the implied resource pattern 0.

A Grid service may be also defined informally as a service that is designed to operate in a Grid environment, and meet the requirements of the Grid(s) in which it participates. The term Grid refers to “a system that is concerned with the integration, virtualization, and management of services and resources in a distributed, heterogeneous environment that supports collections of users and resources (virtual organizations) across traditional administrative and organizational domains (real organizations)”[8].

A GridService -similar to the Service it extends- denotes the set of capabilities which are offered to its consumers as well as the Resources that a consumer should provide to the Grid Service in order to utilize its capabilities. A Grid Service instance can only be invoked if it has been associated to a Resource instance of the required Resource type.

**Attributes**
No attributes

**Associations**
- /protocol:GridServiceInterface [*] (redefines type) Specifies the GridServiceInterface type of the Grid Service. The capabilities and the required resources of the Grid Service should be the ones specified in the GridServiceInterface.
- /requiredResources:Resource [*] Identifies the Resources that should be provided to the Grid Service in order to invoke its capabilities. This association is derived from the Resources required by the GridServiceInterface which defines the type of the Grid Service at hand.

**Constraints**
No Constraints
Semantics
A GridService specifies the capabilities provided by a Participant and the Resources that a consumer must provide in order to utilize its capabilities. A GridService and a GridRequisition are typed elements whose type is specified by a GridServiceInterface. A GridService provides the capabilities that are specified by the associated GridServiceInterfaces and requires the specified Resources for their provision.

Notation
TBS

Additions to UML 2.X

**GridServiceInterface**
Specifies the Interface of a GridRequisition and/or a GridService

Generalization
- ServiceInterface

Description
A GridServiceInterface defines the way to interact with a GridService. It is used as a type or a protocol for a GridService and a GridRequisition similar to the way that a ServiceInterface is used. Further to the features provided by the ServiceInterface, GridServiceInterface addresses the specification of the Resources required for utilizing the capabilities of a GridService.

Attributes
No attributes

Associations
- requiredResource:Resource [*] Identifies the Resources that are required by a GridService in order to grasp its capabilities.

Constraints
No Constraints

Semantics
A GridServiceInterface provides a description of the functionality provided by a GridService, the required Resources along with how one may interact with the service.

Notation
TBS

**Resource**
A resource is a stateful entity that can be either a software component (ranging from a simple file to a software system) or a hardware component (e.g. a lab instrument).

Generalization
- Class

Description
A Resource is a stateful entity which could be either a software or a hardware component. In a Grid context a Resource may be also considered as a set of properties whose values are controlled by a GridService which is associated to that Resource.

A Resource specifies a type whose properties are manipulated by an associated Grid service. It is instantiated, accessed and modified using the implied Resource pattern that has been specified in [8].

Attributes
No attributes

Associations
TBS

Constraints
No Constraints

Semantics
A Resource is a representation of a software or hardware entity whose properties are controlled by an associated Grid service. It is also regarded as a state holder whose state can be accessed and modified using the related stateful services, i.e. Grid services.

Notation
TBS

3.1.2.4. Grid Service Alternative Class Description
In this section we illustrate an alternative representation of the concepts used for the modeling of Grid services. Compared to the model presented in Figure 5 the one presented below uses a different representation for the Resource element. Therefore, in order to avoid repetition, we will only describe the modified concepts.

![Alternative Grid Service model](image)

**Figure 6 Alternative Grid Service model**

*Resource*
A resource is a representation of a software or hardware entity. It comprises a set of properties which map to the entity’s properties.

**Generalization**

- Type

**Description**

A Resource is regarded as a Type which comprises a set of properties. It is considered as a representation of a real entity that can be either a software or hardware component. Its properties can be accessed and controlled by an associated Grid service using the implied resource pattern [8].

A Resource is also regarded as a state holder whose state can be accessed and modified using a related stateful service.

**Attributes**

No attributes

**Associations**

TBS

**Constraints**

No Constraints

**Semantics**

A Resource is a Type which consists of a set of properties. It is a representation of a stateful entity which could be either a software or hardware component. Its properties cannot be directly manipulated; instead they are accessed and managed only via an associate stateful service using the implied resource pattern [8].

**Notation**

TBS

### 3.2. Example

#### 3.2.1. Agent Example

In the following, we demonstrate how to model AIPs the modified UML Sequence Diagrams. The CNP belongs to the family of the cooperation protocols and is the most prominent protocol in distributed artificial intelligence as it provides a solution for the connection problem, i.e., to find an appropriate agent instance to work on a given task. It bases on the contracting mechanism used by business to govern the exchange of goods and services and though uses a minimum of messages which makes it very efficient for task assignment.

In principle, the protocol defines how an initiator sends out a number of calls for proposal to a set of participants. Some of these participants will refuse the call, while others may come up with a proposal. The initiator then evaluates the proposals and accepts the most adequate(s) and rejects the others. For those that are accepted, there are three different final results sent back to the initiator.

To demonstrate how to use Haugen’s approach, we figure out how to model the CNP with the provided extensions.

As said before, in order to distinguish between the different roles, we introduce subsets. Beside the basic roles of “initiator” and “participant”, we further introduce the subsets of “participant” namely “refused”, “rejected” and “accepted” (see Figure 7). These subsets are then used within the protocol as illustrated in Fehler! Verweisquelle konnte nicht gefunden werden.. The subsets “refused”,

Copyright © SHAPE Consortium 2007-2010
"rejected" and "accepted" are subsets of "p". In addition to the subset notation on messages, there exists an extended alt fragment with an iterator-clause. The alt {all} fragment iterates over all participants of the accepted subset meaning that every accepted participant has the choice whether answering with failure, inform-done or inform-result.

Figure 7: Configuration with subsets

Figure 8: The Contract Net Protocol using Subset Notation on messages
3.2.2. Grid and Peer-to-Peer Example

This document provides an extended version of the Purchase Order Process example that has been used earlier in this document, in order to exemplify the need for the utilization of P2P and Grid services. This example aims to establish the foundation for understanding the P2P and Grid service details.

3.2.2.1. Introduction

This example starts by describing the business motivation which establishes the business goals and objectives that are to be achieved as well as the existing IT resources that may be reused by the process. As in the UPMS submission, the example is accompanied by a high-level business process in BPMN that expresses the business organizational and operational requirements.

3.2.2.2. Business Goals and Objectives

A consortium of companies has decided to collaborate in order to produce a reusable service for processing purchase orders. The goals of this project are summarized as:

- Establish a common means of processing purchase orders.
- Ensure orders are processed in a timely manner, and deliver the required goods.
- Help minimize stock on hand.
- Minimize production and shipping costs
- Reuse existing IT infrastructure

**Existing Infrastructure**

The consortium has already deployed a Peer-to-Peer system that interconnects a set of warehouses which are distributed across various geographical locations. This P2P network facilitates the inventory management as well as the communication among all warehouses. Each of the existing warehouses is an independent, autonomous node of the P2P network and has the same capabilities as all the other nodes of the network.

Further to the P2P network which interconnects the warehouses, the consortium owns a Grid infrastructure as well as a set of services providing for the planning of subcontracts and supplies in order to support the current as well as the emerging production plans. These services take as input the current production plans along with the list of materials and tasks that are necessary for the fulfillment of an order. The outcome is a purchase plan and a subcontracting plan that are needed for the completion of the production plan in due time.

3.2.2.3. Business Processes

An abstract description of the business process which satisfies the aforementioned needs as perceived by an analyst from a member company is illustrated in Figure 9. This defines a simple sketch of the business process without the details required to create a services contract that could use strict fulfillment. Figure 10 below shows a more detailed version of the business process.
Figure 10 represents a refinement of the first process that is intended to provide more details on the identified process steps. This model could be used to create strict service contracts.
The purchase order process initiates five parallel activities: one for managing production and shipping scheduling; another for price calculation and invoicing; a third one for shipping the ordered products; a fourth one for performing the inventory checking; and a fifth one for the planning of necessary supplies. Processing starts by initiating a price calculation based on the ordered products. However, this price is not yet complete since the total invoice depends on where the products are produced, the amount of shipping costs and the extra costs which incur from the ordering of missing supplies. At the same time, the order is sent to the inventory in order to determine the necessary materials needed for its fulfillment, i.e. the ones that are currently in stock as well as the ones that are missing and need to be ordered. The production scheduling is postponed until the process determines the supplies that need to be purchased and their related delivery plan. Following that, the process calculates when the products will be available and from which locations. In parallel, the process requests shipping and then waits for a shipping schedule to be sent from production scheduling. Once the production schedule is available, the invoice can be completed, returned to the customer, and the goods are shipped.
3.2.2.4. Business Requirements

The requirements of the business process that has been specified above using BPMN notation can be captured using a Contract which dictates the roles involved in the process as well as their interactions.

The requirements of the Extended Purchase Order process that is illustrated in Figure 11 are the same with the ones specified in the BPMN model illustrated before. The identified roles and their interactions enable us to identify their interfaces, which are illustrated in Figure 12.

The identified requirements, business roles and interfaces leverage the identification of the services that will provide the specified interfaces and will implement the specified roles.
3.2.2.5. Service Identification

Based on the requirements that have been identified, we can specify the contracts for the services that will fulfill them. This will facilitate us in describing the required service interfaces and the necessary roles that satisfy the business needs. In the case of the inventorying and supplying interfaces, since these have already been implemented (these services are part of the existing infrastructure), there is no need to identify their service interfaces again. Nonetheless, in order to ensure that the provided service interfaces accommodate the necessary operations to support the specified business requirements, we specify them in Figure 13.

The identified service contracts facilitate the specification of the roles that each of these contracts will accommodate as well as the interfaces that these roles will support. In the following we illustrate the Interfaces for the inventorying and supplying services.

**Inventory Service Interface**

The inventorying service checks all warehouses to see if there are missing parts that need to be supplied so as to facilitate the timely delivery of an order. The list of missing parts is used by the supplying service in order to schedule their delivery in a timely manner that would facilitate the order's completion.
The inventory contract defines the requirements for the two roles that participate in the specified interaction. As it can be seen in Figure 14, the client utilizes the Inventoring interface provided by the inventoryMngr role.

The identified service interface is similar to the one that is provided by the existing Peer-to-Peer infrastructure which interconnects the warehouses of the consortium. Therefore, there is no need in amending the existing service with new functionality.

**Supplying Service Interface**

The goal of the supplying service is to provide a supply plan for the missing parts of an order. The generated supply plan is utilized by the Scheduling service in order to prepare the production plan. The service contract for the Supplying service could be such as the one presented in Figure 15.

![Figure 15: Supplying Service Contract](image)

Nonetheless, according to the problem specification a supply planning service is already provided as a Grid service over the Grid infrastructure owned by the consortium. Grid services are defined as stateful services which utilize specific resources for their provision. It is thus required that the interface of such a stateful service provides a description of the resource that is needed for its operation. Hence, the service contract which adequately describes the service interface for the Supplying Grid service is presented in Figure 16.

![Figure 16: GridService contract](image)

As it can be seen in Figure 16, the Supplying Grid service interface necessitates the use of a ProductionsPlanDB resource. The latter represents a database holding the current production plans of the consortium. The properties of that resource contain information which is utilized by the Grid service implementing this interface while being executed.
References


3. Case study scenario descriptions and requirements specifications. Deliverable D1.1. SHAPE.


Appendix A: SoaML

The SoaML extends the UML2 to support an explicit service modeling in distributed environments. It makes use of two different extension mechanisms. It provides a UML2 profile for service modeling and a metamodel that merge UML2 with service representation elements.

Services can be used for different purposes on software projects. For example: the can be used to expose functionality for latter use; they can be used to describe extension points in new systems; they can be used to integrate systems or they can be used to specify a service architecture for a given purpose.

Accordingly SoaML is created to support multiple service modeling scenarios taking into account the broad usage of the service technologies. SoaML can be used to describe:

- services exposed by a system
- service architecture through the identification of the expected participants, their services, and their expected interactions
- service contracts between two or more participants
- service usage behavior, how the different operations of the services are sequenced to obtain a desired result
- describe services required by a system to complete its mission

Apart from these core modeling capacities, SoaML also provides some convenience modeling features to facilitate some service modeling scenarios. For example, features to describe capabilities of systems or subsystems to provide service, features to define classification schemes or features to link SoaML models with other business level specifications.

A.1 Implementation

The SoaML specification provides information to implement the metamodel in two ways: as a UML profile and as a metamodel. The implementation as a profile will require a UML2 compliant with the UML2 L3 to make use of collaborations. The implementation as a metamodel requires of components able to edit models that comply with the metamodel and save them in XMI format.

A.2 Metamodel

The focus of the D3.3 is on the metamodel rather than in the profile therefore we will describe the SoaML metamodel, describing the profile based representation of the different elements in the notation section.

A design principle in the development of SoaML was not to create elements to describe things that are possible to be described with current modelling languages. Having that on mind, it was decided to build SoaML over UML2, as UML2 already provides capabilities to describe multiple facets of services such as information or behaviour.

The following figure describes the relationship of SoaML with UML2. As it can be seen in the figure SoaML merge UML2 L3 elements (UML L3 contains collaborations which are relevant for SoaML) and elements coming from the Services package that introduce elements to describe service concepts which are not properly supported by UML2.
Next we will focus in the Service package who provide new elements to UML2 to represent services. The metamodel extends UML2 in five main areas: Participants, ServiceInterfaces, ServiceContracts and Service data. The participants allow to define the service providers and consumers in a system. ServiceInterfaces make it possible to explicitly model the operation provided and required to complete the functionality of a service. The ServiceContracts are used to describe interaction patterns between service entities. Finally, the metamodel also provide elements to model service messages explicitly and also to model message attachments.

Besides, the metamodel also provides some convenience elements to facilitate some modeling scenarios such as capabilities, classifications schemes and milestones.

Figure 18 introduces the elements which support the Participants and the ServiceInterface modeling. A Participant may play the role of service provider, consumer or both. When a participant works as a provider it contains ServicePoints. On the other hand, when a participant works as a consumer it contains RequestPoints.

A ServiceInterface can be used as the protocol for a Service or a Request point. If it is used in a ServicePoint, it means that the Participant who owns the ports is able to implement that ServiceInterface. If it is used in a RequestPoint, it means that the Participant uses that ServiceInterface.

NOTE: Colored Boxes represent elements imported from UML2

Figure 19 presents the elements which support the ServiceContract Modeling. These contracts can later be realized by service elements such as Participants, ServiceInterfaces or Ports (Request or
Service points). Another element included in this figure is the ServiceArchitecture. The ServiceArchitecture is used to model service architectures and their owned behaviors.

![ServiceContracts and ServiceArchitectures](image)

**Figure 19: ServiceContracts and ServiceArchitectures**

Figure 20 presents the elements which support the Data Modeling. Attachments are used to model elements that have their own identity when they are taken out of the system. For example we can define an attachment to send a text document that can be used with other applications external to the system. The MessageType is used to explicitly identify data elements that will travel among the participants of the service interaction.

![Data modeling](image)

**Figure 20: Data modeling**

Figure 21 presents the elements to support capability modeling. Capabilities allows to model service architectures in a “Participant agnostic” way. It allows to focus in the features of the systems without giving to much details on who implement them.

![Capabilities](image)

**Figure 21: Capabilities**

Figure 22 presents the elements to support the definition of milestones. Milestone allows defining progress states on behavioral diagrams. Service interaction can be long lived interactions therefore it could be useful in some scenarios to have elements to support the process state evaluation.
A.1.1 Class Descriptions

A.1.1.1 Agent

**Definition**

An Agent is a classification of autonomous entities that can adapt to and interact with their environment. It describes a set of agent instances that have features, constraints, and semantics in common. Agents in SoaML are also participants, providing and using services.

**Generalizes**

Participant

**Description**

An agent is a kind of Participant with some additional characteristics. Agents are autonomous, interactive and adaptive components. They are capable of acting without direct external intervention. They are able to communicate with the environment and other agents. They can learn and evolve over the time.

**Attributes**

No additional attributes.

**Associations**

No additional associations.

**Constraints**

The property isActive, inherited from class, must always be true.

**Semantics**
Agent extends Participant with the ability to be active, participating components of a system. They are specialized because they have their own thread of control or lifecycle. Another way to think of agents is that they are “active participants” in an SOA system. Participants are Components whose capabilities and needs are static. In contrast, Agents are Participants whose needs and capabilities may change over time.

In SoaML, Agent is a Participant (a subclass of Component). A Participant represents some concrete Component that provides and/or consumes services and is considered an active class (isActive=true). However, SoaML restricts the Participant’s classifier behavior to that of a constructor, not something that is intended to be long-running, or represent an “active” lifecycle. This is typical of most Web Services implementations as reflected in WS-* and SCA.

Agents possess the capability to have services and Requests and can have internal structure and ports. They collaborate and interact with their environment. An Agent's classifierBehavior, if any, is treated as its life-cycle, or what defines its emergent or adaptive behavior.

A.1.1.1.8 Notation
An agent with the profile can be designated using the Component notation including the <<agent>> keyword. Using the metamodel specific notation the way to designate the agent will be similar.

A.1.1.2 Attachment

A.1.1.2.1 Definition
A part of a Message that is attached to rather than contained in the message.

A.1.1.2.2 Generalizes
Property.

A.1.1.2.3 Description
The objective of Attachment is to allow the definition of elements in the messages that have their own identity and which are separable from the main message. Examples of attachments can be images, word documents, pdf documents, or music samples.

Attachments are usually created by external applications and are expected to be processed by external applications.

A.1.1.2.4 Attributes
encoding: <Primitive Type> String – It can be used to provide information on the codification of the attachment in the message

A.1.1.2.5 Associations
No additional associations.
A.1.1.2.6 Constraints
No additional constrains.

A.1.1.2.7 Semantics
An Attachment extends Property to distinguish attachments owned by a MessageType from other
ownedAttributes. The ownedAttributes of a MessageType must be either PrimitiveType or
MessageType. The encoding of the information in a MessageType is specified by the encoding
attribute. In distributed I.T. systems, it is often necessary to exchange large opaque documents in
service data in order to support efficient data interchange. An Attachment allows portions of the
information in a MessageType to be separated out and to have their own encoding and MIME type,
and possibly interchanged on demand.

A.1.1.2.8 Notation
An attachment with the profile can be designated using the property notation including the
<<attachment>> keyword. Using the metamodel specific notation the way to designate the attachment
will be similar.

A.1.1.3 Capability

Figure 25: Capability

A.1.1.3.1 Definition
A Capability is the ability to act and produce an outcome that achieves a result. It can. Specify a
general capability of a participant as well as the specific ability to provide a service.

A.1.1.3.2 Generalizes
Class

A.1.1.3.3 Description
A Capability models the ability to act and produce an outcome that achieves a result that may provide
a service specified by a ServiceContract or ServiceInterface irrespective of the Participant that might
provide that service. A ServiceContract, alone, has no dependencies or expectation of how the
capability is realized – thereby separating the concerns of “what” vs. “how”. The Capability may
specify dependencies or internal process to detail how that capability is provided including
dependencies on other Capabilities. Capabilities are shown in context using a service dependencies
diagram.

A.1.1.3.4 Attributes
No additional attributes.

A.1.1.3.5 Associations
No additional associations.
A.1.1.3.6 Constraints
No additional constrains.

A.1.1.3.7 Semantics
A Capability identifies or specifies a cohesive set of functions or capabilities that a service provided by one or more participants might offer. Capabilities are used to identify needed services, and to organize them into catalogues in order to communicate the needs and capabilities of a service area, whether that be business or technology focused, prior to allocating those services to particular Participants. For example, service capabilities could be organized into UML Packages to describe capabilities in some business competency or functional area. Capabilities can have usage dependencies with other Capabilities to show how these capabilities are related. Capabilities can realize ServiceInterface and so specify how those ServiceInterfaces are supported by a Participant. Capabilities can also be organized into architectural layers to support separation of concern within the resulting service architecture.

Each capability may have owned behaviors that are methods of its provided Operations. These methods would be used to specify how the service capabilities might be implemented, and to identify other needed service capabilities.

A.1.1.3.8 Notation
A capability with the profile can be designated using the Class notation including the <<capability>> keyword. Using the metamodel specific notation the way to designate the capability will be similar.

A.1.1.4 Collaboration

A.1.1.4.1 Definition
This is an auxiliary element to extend the UML collaboration with the isStrict attribute. This element is latterly extended by the ServiceContract and the ServiceArchitecture which are described in more detail.

A.1.1.4.2 Generalizes
Collaboration

A.1.1.4.3 Description
Collaboration describes a ServiceContract or a ServiceArchitecture among several parts. Parts indicate the different roles in the collaborations. The collaboration defines an interation pattern that can be applied to a set of participants through a CollaborationUse.

A.1.1.4.4 Attributes
isStrict: <Primitive Type> Boolean – This attribute provides an indication of the level of adhesion to the behavior described in the Collaboration by the elements that play the roles contained in it.

A.1.1.4.5 Associations
No additional associations.

**A.1.1.4.6 Constraints**

No additional constrains.

**A.1.1.4.7 Semantics**

A Collaboration is a statement about an interaction pattern between a set of classifiers. It can define the roles acting in the interaction, the communication channels among the different acting partners, and the expected behaviour of those roles.

When instantiating the collaboration, through a collaboration use, those participants with a rolebinding association with a role are expected to comply with the expected behaviour of that role in the collaboration.

The ability of a containing Classifier to provide or use capabilities, have structure, or behave in a manner consistent with that expressed in its Collaboration type. It is an assertion about the structure and behavior of the containing classifier and the suitability of its parts to play roles for a specific purpose.

If the Collaboration has isStrict=true, then their collaboration usages will have the isStrict=true.

**A.1.1.4.8 Notation**

No additional notation, we what for it to look like a regular collaboration plus one attribute.

**A.1.1.5 CollaborationUse**

![CollaborationUse](image)

**Figure 27: CollaborationUse**

**A.1.1.5.1 Definition**

CollaborationUse is extended to indicate whether the role to part bindings are strickly enforced or loose.

**A.1.1.5.2 Generalizes**

CollaborationUse

**A.1.1.5.3 Description**

A CollaborationUse explicitly indicates the ability of an owning Classifier to fulfill a ServiceContract or adhere to a ServicesArchitecture. A Classifier may contain any number of CollaborationUses which indicate what it fulfills. The CollaborationUse has roleBindings that indicate what role each part in the owning Classifier plays. If the CollaborationUse is strict, then the parts must be compatible with the roles they are bound to, and the owning Classifier must have behaviors that are behaviorally compatible with the ownedBehavior of the CollaborationUse’s Collaboration type.

Note that as a ServiceContract is binding on the ServiceInterfaces named in that contract, a CollaborationUse is not required if the types are compatible.

**A.1.1.5.4 Attributes**

isStrict: <Primitive Type> Boolean – This attribute provides an indication of the level of adhesion to the behavior described in the Collaboration by the elements that play the roles contained in it.
A.1.1.5.5 Associations
No additional associations.

A.1.1.5.6 Constraints
No additional constrains.

A.1.1.5.7 Semantics
A CollaborationUse is a statement about the ability of a containing Classifier to provide or use capabilities, have structure, or behave in a manner consistent with that expressed in its Collaboration type. It is an assertion about the structure and behavior of the containing classifier and the suitability of its parts to play roles for a specific purpose.

A CollaborationUse contains roleBindings that binds each of the roles of its Collaboration to a part of the containing Classifier. If the CollaborationUse has isStrict=true, then the parts must be compatible with the roles they are bound to. For parts to be compatible with a role, one of the following must be true:
- The role and part have the same type,
- The part has a type that specializes the type of the role,
- The part has a type that realizes the type of the role, or
- The part has a type that contains at least the ownedAttributes and ownedOperations of the role. In general this is a special case of item 3 where the part has an Interface type that realizes another Interface.

A.1.1.5.8 Notation
No additional notation, we what for it to look like a regular collaborationUse plus one attribute.

A.1.1.6 Port

A.1.1.6.1 Definition
Extends UML Port with a means to indicate whether a Connection is required on this Port or not.

A.1.1.6.2 Generalizes
Port

A.1.1.6.3 Description
Request and Service points are both Ports from which information is exchanged with other external entities. The Ports are not always mandatory and it is possible to specify optional Ports using the connectorRequired attribute.

Participants can include one or more Ports to identify ports as service interaction points. The places from which the services are exchanges can be Services or Requests. Services are Ports that describe the services provided by the Participant, while Requests are Ports that describe the services that the Participant must consume in order to be able to provide their Services.

(Attr connectorRequired ) Port provide to Services and Requests with the Port attribute to indicate whether a connector is required on this connection point. This enables the designer to specify optional services and requests that are not critical for the participant.
A.1.1.6.4 Attributes
connectorRequired: <Primitive Type> Boolean – If the value is true, there must be a connector attached to the Port.

A.1.1.6.5 Associations
No additional associations.

A.1.1.6.6 Constraints
No additional constrains.

A.1.1.6.7 Semantics
(Attr connectorRequired) When a Port, a Service or a request, has the attribute connectorRequired is set to true (the default value) this implies that there must be at least one connector to that Port. This means that if the value is true the Service must be consumed or the request must be satisfied.

On the other hand, when the value of connectorRequired is set to false this implies that the Participant is able to work without that Service or request. This may have some implication on the service quality that the Participant should be able to manage.

(ServiceInterface) The Port inherits from UML TypedElement, therefore it has a type attribute. The type of the Port determines the capabilities to be provided or requested through it. But the assignment of the type is optional. Therefore, if it is not assigned it means that the specification of the Service or Request may include or not the specification of the interfaces, operations and behaviours to be provided through the Port.

(Interface/ServiceInterface) A Port can be typed with interfaces. It is possible to use regular UML Interfaces or ServiceInterfaces defined in this metamodel. If UML Interfaces are used, it implies that the Participant that owns the Port requires that capabilities from external entities, or provides capabilities to external entities. If ServiceInterfaces are used, it means that it will be possible to associate to that Port one or more required and provided Interfaces between the provider and the consumer. The sequence of usage of operations of the different interfaces can be described by the ServiceInterface.

(Participant) When a Port (Service or request) is attached to a Participant this means that the Participant is able to provide and process all the interface or the ServiceInterface associated to the Port type. It also means that the Participant is able to cope with the roles of the Contracts to which the Port is associated through a Role Binding.

A.1.1.6.8 Notation
No additional notation, we what for it to look like a regular port plus one attribute.

A.1.1.6.9 Examples
A.1.1.7 MessageType

**A.1.1.7.1 Definition**
The specification of information exchanged between service consumers and providers.

**A.1.1.7.2 Generalizes**
DataType

**A.1.1.7.3 Description**
A MessageType is a kind of value object that represents information exchanged between participant requests and services. This information consists of data passed into, and/or returned from the
invocation of an operation or event signal defined in a service interface. A MessageType is in the domain or service-specific content and does not include header or other implementation or protocol-specific information. MessageTypes can be used to describe input, output and exception messages. Note: MessageType should generally only be applied to DataType since it is intended have no identity. However, it is recognized that many existing models do not clearly distinguish identity, either mixing Class and DataType, or only using Class. Recognizing this, SoaML allows MessageType to be applied to Class as well as DataType. In this case, the identity implied by the Class is not considered in the MessageType. The Class is treated as if it were a DataType.

A.1.1.7.4 Attributes
encoding: String [0..1] Specifies the encoding of the message payload.

A.1.1.7.5 Associations
No additional associations.

A.1.1.7.6 Constraints
- MessageType cannot contain ownedOperations.
- MessageType cannot contain ownedBehaviors.
- All ownedAttributes must be Public
- All ownedAttributes of a MessageType must be PrimitiveType, DataType or another MessageType or a reference to one of these types.

A.1.1.7.7 Semantics
A service Operation is any Operation of an Interface provided or required by a ServicePoint or RequestPoint. Service Operations may use two different parameter styles, document centered (or message centered) or RPC (Remote Procedure Call) centered. Document centered parameter style uses MessageType for ownedParameter types, and the Operation can have at most one in, one out, and one exception parameter (an out parameter with isException set to true). All parameters of such an operation must be typed by a MessageType. For RPC style operations, a service Operation may have any number of in, inout and out parameters, and may have a return parameter as in UML2. In this case, the parameter types are restricted to PrimitiveType or DataType. This ensures no service Operation makes any assumptions about the identity or location of any of its parameters. All service Operations use call-by-value semantics in which the ownedParameters are value objects or data transfer objects.

A.1.1.7.8 Notation
An MessageType with the profile can be designated using the Class or DataType notation including the <<messageType>> keyword. Using the metamodel specific notation the way to designate the MessageType will be similar.
A.1.1.8 Milestone

**A.1.1.8.1 Definition**
A Milestone is a means for depicting progress in behaviors in order to analyze liveness. Milestones are particularly useful for behaviors that are long or even infinite.

**A.1.1.8.2 Generalizes**
Comment

**A.1.1.8.3 Description**
A milestone depicts progress by defining a signal that is sent to an imaginary observer. The signal contains an integer value that intuitively represents the amount of progress that has been achieved when passing a point attached to this milestone.

**A.1.1.8.4 Attributes**
progress: Integer The progress measurement.

**A.1.1.8.5 Associations**
signal: Signal [0..1] A Signal associated with this Milestone
value: Expression [0..1] Arguments of the signal when the Milestone is reached.

**A.1.1.8.6 Constraints**
No additional constrains.

**A.1.1.8.7 Semantics**
The milestones are used to annotate certain points in the behavioral descriptions of the services. They can be used in the behavioral description of the ServiceArchitectures, ServicesContracts, Participants and ServiceInterfaces.

The Milestone can be understood as a declaration to produce a signal at a specific point in a process. The signal is sent to an observer each time that a point connected to the Milestone is passed during execution. The implementation of the Milestones in the final execution environment is not mandatory, it is only advisable in those cases where there is an interest on monitoring the progress of the execution of certain behaviors.
A.1.1.8.8 Notation

A Milestone with the profile can be designated using the Comment notation including the <<milestone>> keyword. Using the metamodel specific notation the way to designate the Milestone will be similar.

A.1.1.9 Participant

Figure 33: Participant

A.1.1.9.1 Definition

The type of a provider and/or consumer of services. In the business domain a participant may be a person, organization or system. In the systems domain a participant may be a system, or component.

A.1.1.9.2 Generalizes

Component

A.1.1.9.3 Description

A Participant Figure 34 is a special kind of UML Component or Class that provides or consumes services. It represents an element of the system, or the system itself, that interacts with other entities outside the boundaries of the system.

Participants are used to distinguish regular elements (Components and Classes) from those that maintain service relationships with entities outside the boundaries of the participant. For example they can be used in the design of a supply chain management system to differentiate those parts, or components, that exchange information with other organizations. Participants can also be used to identify the different organizations taking part in complex business transactions.

A.1.1.9.4 Attributes

No additional attributes.

A.1.1.9.5 Associations

requisitions: <Class> Request – Derived attribute that associates the Request ports contained by the Participant

services: <Class> Service – Derived attribute that associates the Service ports contained by the Participant

A.1.1.9.6 Constraints

- A Participant cannot realize or use Interfaces directly, it must do so through Service and Request points.
- Note that the technology implementation of a component implementing a participant is not bound by the above rule in the case of it’s internal technology implementation, the connections to a participant components “container” and other implementation components may or may not use services.
A.1.1.9.7 Semantics

As a UML class or Component the participant represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable. In the same way that components can, the Participant can package other UML elements to support its definition and to organize the model.

Participants may act as service consumer, providers or both. A participant will act as pure consumer if it only owns Requisitions and no Services (Figure 35). On the other hand it will act as a pure provider if it only owns Services and no Requisitions (Figure 36).

(Services/Requests) When a ConnectableElement (Service or Request) port is added to a Participant, this implies that the Participant is able to provide or consume the capabilities defined that ConnectableElement. The capabilities can be specified Interface or ServiceInterface type of the ConnectableElement, or by the Contracts supported by that ConnectableElement. It is important to remark that when the Participant Services a ServiceInterface (or regular UML Interface) that means that the Participant is able to provide the provided interfaces and consume the required interfaces (This is because the Participant is acting as the Provider of the ServiceInterface). The situation is different when a Participant Request a ServiceInterface. In that case that means that the Participant is able to consume the provided Interfaces and provide the required Interfaces (This is because the Participant is acting as the consumer of the ServiceInterface).

(ServiceInterfaces) The Interfaces or the ServiceInterfaces implemented by the Participant are referenced through ConnectableElements (Services or Requisitions). When a ServiceInterface is pointed through a ConnectableElement that means that the Participant must implement the operations of the ServiceInterface and must comply with the behavior of the ServiceInterface.

(Contracts) Participants can indicate the service contracts they fulfill and the roles they implement on those contracts. There are several ways to specify this relationship. It can be done in the context of a service architecture or it can be done in the context of a participant. The implication of this binding is that the participant should, or must (in case the Contract is Strict), implement all the responsibilities specified in the contract for the linked role. For example, if the role type is a service interface, the participant should (or must) provide that service interface complying with the behavior of the contract. If the role type is a participant, this participant should (or must) provide all the features of that participant complying with the behavior of the contract.

Participant can include behavioral descriptions to describe the way in which the different Services and Requisitions are executed to complete the objective of the Participant. This specification does not constrain the way to describe the behavior of the component. Any UML behavioral specification method can be used to describe the activity of the Participant. This includes, but not is not limited to, activities, interactions, state machines, protocol state machines, and/or opaque behaviors.

A.1.1.9.8 Notation

A participant with the profile can be designated using the Component notation including the <<participant>> keyword. Using the metamodel specific notation the way to designate the Participant will be similar.

A.1.1.9.9 Examples

Figure 34: Diagram 01. Simple
Figure 35: Diagram 02. Provider

Figure 36: Diagram 03. Consumer

Figure 37: Diagram 04. ContractFulfillServiceArchitecture

Figure 38: Diagram 05. ContractFulfillParticipant
Figure 39: Diagram 06. Behavior

Figure 40: Diagram 07. ComposedParticipant
Figure 41: Diagram 08. SpecificationImplementation

Figure 42: Diagram 09. BehaviorOperation

A.1.1.10 ParticipantArchitecture

A.1.1.10.1 Definition
The high-level services architecture of a participant that defines how a set of internal and external participants use services to implement the responsibilities of the participant.

A.1.1.10.2 Generalizes
Class

A.1.1.10.3 Description
A ParticipantArchitecture describes how internal participants work together for a purpose by providing and using services expressed as service contracts. The participant architecture is a kind of services architecture (See ServicesArchitecture, below) for a particular participant. By expressing the use of services, the ParticipantArchitecture implies some degree of knowledge of the dependencies between the participants in the context of the containing participant.

A participant architecture is similar to a ServicesArchitecture detailed below and the description of such an architecture is not repeated here. The only different is that a participant architecture is based on a structured classifier rather than a collaboration and can therefore for external ports that represent interactions with external participants.
A Participant may play a role in any number of services architecture thereby representing the role a participant plays and the requirements that each role places on the participant.

A.1.1.10.4 Attributes
No additional attributes.

A.1.1.10.5 Associations
No additional associations.

A.1.1.10.6 Constraints
- The parts of a ParticipantArchitecture must be typed by a Participant or Capability. Each participant satisfying roles in a ServicesArchitecture or ParticipantArchitecture shall have a port for each role binding attached to that participant. This port shall have a type compliant with the type of the role used in the ServiceContract.

A.1.1.10.7 Semantics
Standard UML2 Collaboration semantics are augmented with the requirement that each participant used in a services architecture must have a port compliant with the ServiceContracts the participant provides or uses, which is modeled as a role binding to the use of a service contract.

A.1.1.10.8 Notation
A ParticipantArchitecture with the profile can be designated using the Class notation including the <<participantArchitecture>> keyword. Using the metamodel specific notation the way to designate the ParticipantArchitecture will be similar.

A.1.1.11 Property

Figure 44: Property

A.1.1.11.1 Definition
The Property class augments the standard UML Property with the ability to be distinguished as an identifying property meaning the property can be used to distinguish instances of the containing Classifier. This is also known as a “primary key”.

A.1.1.11.2 Generalizes
Property

A.1.1.11.3 Description
The objective of this specialization of the UML2 Property element is to provide the mechanisms to identify instances of the containing classifier in distributed systems.

A.1.1.11.4 Attributes
isID: <Primitive Type> Boolean – Indicates that the property can be used to identify the instances of the contained classifier.

A.1.1.11.5 Associations
No additional associations.

A.1.1.11.6 Constraints
No additional constrains.

A.1.1.11.7 Semantics
When the isID attribute is set to true that implies that then values assigned to that property should be unique in that system. These properties can be used to distinguish instances of the containing classifier.

A.1.1.11.8 Notation
No additional notation, we what for it to look like a regular collaboration plus one attribute.

A.1.1.12 RequestPoint

Figure 45: RequestPoint

A.1.1.12.1 Definition
A request models the use of a service by a participant and defines the connection point through which a Participant makes requests and uses or consumes services.

A.1.1.12.2 Generalizes
Port

A.1.1.12.3 Description
The objective of the RequestPoint is to make it possible to distinguish the main service or services of the Participant and those who are required in order to be able to provide the main services. For example a Purchasing company main service is the PurchaseService, but in order to provide that service it may require other external services such as Invoicing, Scheduling or Shipping services.

A.1.1.12.4 Attributes
No additional attributes.

A.1.1.12.5 Associations
No additional associations.

A.1.1.12.6 Constraints
No additional constrains.

A.1.1.12.7 Semantics
A Request inherits from Port. It is a Port that can provide and request operations to external entities. That means that the Participant who owns the request is action as the consumer of the services.

More concretely, if the type of the Request is an Interface, the Participant must be able to consume that interface. In that case the Interface will be required by the Port. If the type of the Request is a ServiceInterface, it means that the Interfaces realized by the ServiceInterface will be required by the Port, and the Interfaces used by the ServiceInterface will be provided by the Port. It is important to note that this semantic differs from the standard UML semantics for port and its types.

A.1.1.12.8 Notation

A RequestPoint with the profile can be designated using the Class notation including the <<requestPoint>> keyword. Using the metamodel specific notation the way to designate the RequestPoint will be similar.

A.1.1.13 ServicePoint

Figure 46: ServicePoint

A.1.1.13.1 Definition

A ServicePoint is a capability offered by one entity or entities to others using well defined terms, conditions and interfaces.

A.1.1.13.2 Generalizes

Port

A.1.1.13.3 Description

The objective of the ServicePoint is to make it possible to distinguish the main service or services of the Participant and those who are required in order to be able to provide the main services. For example a Purchasing company main service is the PurchaseService, but in order to provide that service it may require other external services such as Invoicing, Scheduling or Shipping services.

A.1.1.13.4 Attributes

No additional attributes.

A.1.1.13.5 Associations

No additional associations.

A.1.1.13.6 Constraints

No additional constrains.

A.1.1.13.7 Semantics

A Service inherits from Port. It is a Port that can provide and request operations to external entities. From a conceptual point of view it identifies the interfaces that constitute the objective of the Participant.
More concretely, if the type of the Service is an Interface, the Participant must be able to provide that interface. In that case the Interface will be provided by the Port. If the type of the Service is a ServiceInterface, it means that the Interfaces realized by the ServiceInterface will be provided by the Port, and the Interfaces used by the ServiceInterface will be required by the Port.

A.1.1.13.8 Notation
A ServicePoint with the profile can be designated using the Class notation including the <<servicePoint>> keyword. Using the metamodel specific notation the way to designate the ServicePoint will be similar.

A.1.1.14 ServiceChannel

A.1.1.14.1 Definition
A communication path between ServicePoints and RequestPoints within an architecture

A.1.1.14.2 Generalizes
Connector

A.1.1.14.3 Description
A ServiceChannel provides a communication path between consumer Requests and provider services

A.1.1.14.4 Attributes
No additional attributes.

A.1.1.14.5 Associations
No additional associations.

A.1.1.14.6 Constraints
- One end of a ServiceChannel must be a RequestPoint and the other a ServicePoint in an architecture.
- The Request and Service connected by a ServiceChannel must be compatible.
- The contract Behavior for a ServiceChannel must be compatible with any protocols specified for the connected requests and Services.

A.1.1.14.7 Semantics
A ServiceChannel is used to connect Requests of consumer Participants to Services of provider Participants at the ServiceChannel ends. A ServiceChannel enables communication between the Request and service.
A RequestPoint specifies a Participant’s needs. A ServicePoint specifies a Participant’s services offered. The type of a RequestPoint or ServicePoint is a ServiceInterface or Interface that defines the needs and capabilities accessed by a Request through Service, and the protocols for using them. Loosely coupled systems imply that services should be designed with little or no knowledge about particular consumers. Consumers may have a very different view of what to do with a service based
on what they are trying to accomplish. For example, a guitar can make a pretty effective paddle if that’s all you have and you’re stuck up a creek without one.

Loose coupling allows reuse in different contexts, reduces the effect of change, and is the key enabler of agile solutions through an SOA. In services models, ServiceChannels connect consumers and providers and therefore define the coupling in the system. They isolate the dependencies between consuming and providing participants to particular Request/service interaction points. However, for services to be used properly, and for Requests to be fully satisfied, Requests must be connected to compatible Services. This does not mean the Request Point and Service Point must have the same type, or that their ServiceInterfaces must be derived from some agreed upon ServiceContract as this could create additional coupling between the consumer and provider. Such coupling would for example make it more difficult for a service to evolve to meet needs of other consumers, to satisfy different contracts, or to support different versions of the same request without changing the service it is connected to.

Loosely coupled systems therefore require flexible compatibility across ServiceChannels. Compatibility can be established using UML2 specialization/generalization or realization rules. However, specialization/generalization, and to a lesser extent realization, are often impractical in environments where the classifiers are not all owned by the same organization. Both specialization and realization represent significant coupling between subclasses and realizing classifiers. If a superclass or realized class changes, then all the subclasses also automatically change while realizing classes must be examined to see if change is needed. This may be very undesirable if the subclasses are owned by another organization that is not in a position to synchronize its changes with the providers of other classifiers.

A RequestPoint is compatible with, and may be connected to a ServicePoint through a ServiceChannel if:

- The Request and Service have the same type, either an Interface or ServiceInterface
- The type of the Service is a specialization or realization of the type of the Request.
- The Request and Service have compatible needs and capabilities respectively. This means the Service must provide an Operation for every Operation used through the Request, the Request must provide an Operation for every Operation used through the Service, and the protocols for how the capabilities are compatible between the Request and Service.
- Any of the above are true for a subset of a ServiceInterface as defined by a port on that service interface.

A.1.14.8 Notation

A ServiceChannel with the profile can be designated using the Connector notation including the <<serviceChannel>> keyword. Using the metamodel specific notation the way to designate the ServiceChannel will be similar.

A.1.15 ServiceContract

A ServiceContract is the formalization of a binding exchange of information, goods, or obligations between parties defining a service.
A.1.1.15.2 Generalizes

Collaboration

A.1.1.15.3 Description

A ServiceContract formalizes the roles and responsibilities for providing and consuming a service, including the information, obligations and items that move between the participants in the realization of that service.

ServiceContracts are used to specify how a service is provided and consumed based on interactions and behaviors involving multiple participants. At minimum, they specify the operations involved in the interaction and the sequence of invocation of those operations.

Participants and Ports can be bound to the different roles of the ServiceContracts. These bindings imply the capability of the Participant or the Ports to fulfill that role.

A.1.1.15.4 Attributes

No additional attributes.

A.1.1.15.5 Associations

No additional associations.

A.1.1.15.6 Constraints

No additional constrains.

A.1.1.15.7 Semantics

As a UML Collaboration the ServiceContract purpose is to describe the interaction among a set of roles. As a UML Collaboration the ServiceContract can also be used to realize UseCases that capture contract requirements.

The role types in a ServiceContract are expected to be Interfaces or ServiceInterfaces (. RolesTypes).

A ServiceContract can also nest in other ServiceContracts. This means that the realisation of the upper level ServiceContract forces the realization of the nested ServiceContracts.

Where the ServiceInterfaces are used as the types of parts in a service contract, the behavior of the serviceInterface must comply with the ServiceContract. However, common practice would be to specify a behavior in the ServiceContract or ServiceInterface not both.

UML Connectors can be used to indicate possible interactions between roles.

A.1.1.15.8 Notation

A ServiceContract with the profile can be designated using the Collaboration notation including the <<ServiceContract>> keyword. Using the metamodel specific notation the way to designate the ServiceContract will be similar.

A.1.1.15.9 Examples

```
<<ServiceContract>>

ACMEPurchaseStandard
```

Figure 49: Diagram 01. Simple
Figure 50: Diagram 02. Roles

Figure 51: Diagram 03. UseCaseRealization

Figure 52: Diagram 04. Connectors

Figure 53: Diagram 05. ConstraintsAndObjectives
A.1.1.16 ServiceInterface

A.1.1.16.1 Definition
The ServiceInterface describe the operations used between a serviceProvide and a service Consumer from the perspective of the provider.

A.1.1.16.2 Generalizes
Class

A.1.1.16.3 Description
ServiceInterface are used to specify the operations involved in a Service interaction. It can also be used to specify the order in which those operations are executed. ServiceInterfaces are used as a type of a Service or Request. A ServiceInterface can require the provision and requisition of operations, therefore a ServiceInterface can imply the implementation and usage of one or more regular UML Interfaces between the service provider and the service consumer.

A.1.1.16.4 Attributes
No additional attributes.

A.1.1.16.5 Associations
No additional associations.

A.1.1.16.6 Constraints
No additional constrains.

A.1.1.16.7 Semantics
As a UML Class the ServiceInterfaces can realize and use other interfaces, this allows to represent complex services that require the bidirectional exchange of information between the provider and the consumer. The ServiceInterfaces are not intended to implement the behavior and the operations they define. ServiceInterfaces are used for specification purposes, the realization of the operations and the behavior is implemented by the concrete Participants.

ServiceInterfaces do not directly contain operations. The operations provided by the ServiceInterface are described in the operations of the realized interfaces. While the consumed interfaces are described in the operations of the used interfaces. Using UML mechanisms operations can be further described with pre and post conditions, parameters and exceptions.

ServiceInterfaces can contain behavioral descriptions to describe the way in which the different provided and required operations are executed to fulfill the objectives of the service.

A.1.1.16.8 Notation
A ServiceInterface with the profile can be designated using the Class notation including the \<<ServiceInterface>> keyword. Using the metamodel specific notation the way to designate the ServiceInterface will be similar.

### A.1.1.17 ServicesArchitecture

![Figure 55: ServicesArchitecture](image)

**A.1.1.17.1 Definition**

The high-level view of a Service Oriented Architecture that defines how a set of participants works together, forming a community, for some purpose by providing and using services.

**A.1.1.17.2 Generalizes**

Collaboration

**A.1.1.17.3 Description**

ServicesArchitectures are used to conceptually relate sets of Participants that work together providing and consuming services for some purpose. ServicesArchitectures can be used to identify the Participants that work together with a common purpose and the Contracts that rule their interactions. ServicesArchitectures can be also use to group all the Participants that belong to a given domain or all the participants that belong to the same organization and relate there service relationships.

A ServicesArchitecture may be specified externally – in a “B2B” type collaboration where there is no controlling entity or as the ServicesArchitecture of a participant - under the control of a specific entity and/or business process.

A Participant may play a role in any number of services architecture thereby representing the role a participant plays and the requirements that each role places on the participant.

**A.1.1.17.4 Attributes**

No additional attributes.

**A.1.1.17.5 Associations**

No additional associations.

**A.1.1.17.6 Constraints**

- The parts of a ServicesArchitecture must be typed by a Participant or capability. Each participant satisfying roles in a ServicesArchitecture shall have a port for each role binding attached to that participant. This port shall have a type compliant with the type of the role used in the ServiceContract.

**A.1.1.17.7 Semantics**

Standard UML2 Collaboration semantics are augmented with the requirement that each participant used in a services architecture must have a port compliant with the ServiceContracts the participant provides or uses, which is modeled as a role binding to the use of a service contract.
A.1.1.17.8 Notation

A ServiceArchitecture with the profile can be designated using the Component notation including the <<ServiceArchitecture>> keyword. Using the metamodel specific notation the way to designate the ServiceArchitecture will be similar.

A.1.1.17.9 Examples

![Diagram 01. Participants](image)

**Figure 56: Diagram 01. Participants**

![Diagram 02. ContractsParticipants](image)

**Figure 57: Diagram 02. ContractsParticipants**
Figure 58: Diagram 03. ConnectableElement

Figure 59: Diagram 04. ContractsConnectableElement

Figure 60: Diagram 05. ParticipantsComponent
Figure 61: Diagram 06. ParticipantsCollaboration
Appendix B: SoaML Profile

The SoaML Profile is provided as a separate document as it was presented in the OMG.