UNIVERSITÄT AUGSBURG

A UML profile and transformation rules for semantic web services

Florian Lautenbacher

Report 2006-20 September 2006

INSTITUT FÜR INFORMATIK

D-86135 AUGSBURG

Table of contents

Table of figures:

A UML profile and transformation rules for semantic web services

Florian Lautenbacher

Programming Distributed Systems
Institute of Computer Science, University of Augsburg, Germany
`lautenbacher@informatik.uni-augsburg.de`

Abstract. This report describes a meta-model and UML profile for modeling semantic web services and the automatic generation of code. It includes a detailed comparison of current approaches, defines an overall meta-model considering all different semantic web service standards and describes transformation rules as well in an informal way as using the openArchitectureWare language XPand. The profile has been integrated into the tool-suite MID innovatorAOX where we modeled the well-known example CongoBuy which was first introduced in the OWL-S standard.

Keywords: UML, MDA, Semantic Web Services, OWL-S, WSMO, WSDL-S, SWSF, ODM, openArchitectureWare, XPand.

1 Introduction

As the number of available web services is steadily increasing, the interest companies have in discovering web services and having an automatic composition is getting higher, too. Several organizations have already proposed web service composition and modeling languages (the most prominent one is WSBPEL [14], but there are many others like XPDL [15] for business-oriented workflows, BPML [16], BPMN [17], etc.), but these standards lack a semantic description of services which can be automatically interpreted by machines and can not be used to make an automatic discovery or composition. To solve these issues several organizations and companies focus currently on the development of semantic-enabled web service technologies. Starting with OWL-S [6] several approaches have been submitted to the World Wide Web Consortium (W3C) such as WSMO [6], WSDL-S [7] and SWSF [9], – each one focusing on different aspects as the other ones. As the number of "standards" is getting higher, the need for an independent way of describing semantic web services increases, too, because the learning curve for the average service developer can be steep for such languages.

The leading organization for object-oriented programming, the Object Management Group (OMG), promotes the model-driven architecture (MDA, [4]) approach towards the analysis, design and implementation of systems. MDA provides an open, vendor-neutral approach to the challenge of business and technology change. It consists of several layers (mostly cited as *CIM*, *PIM* and *PSM*) to specify a system independently of computation, independent of the platform that supports it, specifying the platform itself, choosing a particular platform for the system and transforming the system specification into one for a particular platform and finally into executable code. The primary goals of MDA are portability, interoperability and reusability through an architectural separation of concerns between the specification and implementation of software. One of the key aspects is the usage of the Unified Modeling Language (UML, [2]) to model all kind of aspects. UML (currently version 2.1 is under development) is established in the section of computer science, but also easily understandable for business experts (e.g. business process models as activity diagrams, see e.g. [28]) and supports with its well-defined meta-model (building on MOF, [3]) model transformation and code generation.

To fulfill the need of an independent way of describing semantic web services we developed a meta-model and UML-profile for semantic web services. The advantage of using a UML profile is clear: one can model a service now and generate code using one of the existing standards and if it is foreseeable that another standard becomes accepted one can simply generate code using the other language, too. Our profile should serve the following requirements:

- easy-to-understand and easy-to-use
- be compatible with current standards
- enable code generation
- include the modeling of ontologies (and rules)

Additionally we developed informal transformation rules to generate code from the meta-model and implemented these rules using the openArchitectureWare-language XPand [13]. Our UML-profile has been implemented in the UML tool-suite innovatorAOX 2006 [12]. InnovatorAOX 2006 is developed by MID Enterprise Software Solutions GmbH, Nuremberg, and provides a holistic standard tool environment for object- and function-oriented software development as well as business process and data modeling to help customers solve costly and risk-ridden problems that arise from inadequate tools, software production processes and operational systems management.

This report is organized as follows: In the next chapter we describe the already existing meta-models which we use and give a short explanation about each standard, after that we introduce our approach in chapter 3, define a meta-model and show in chapter 4 how it can be mapped to UML. In chapter 5 we show a short example how the profile can be used. Then we define an informal description for transformation rules (chapter 6), before we specify our transformation rules using the XPand language of openArchitectureWare in chapter 7. Related work, other profiles and their shortcomings are described in chapter 8. The last chapter gives a short outlook to possible extensions and concludes.

2 Comparison and short revision of existing standards

To realize our meta-model we build on existing standards which are established or are likely to establish in the near future. First, we will give a short introduction into some parts of UML, after that we will describe the current semantic web service approaches named OWL-S, WSMO, WSDL-S and SWSF (which have all been submitted to the W3C) before we show shortly two other semantic web service approaches and conclude this chapter with a short summary of the ontology definition meta-model (ODM). Figure 1 shows how the semantic web service submissions can be categorized.

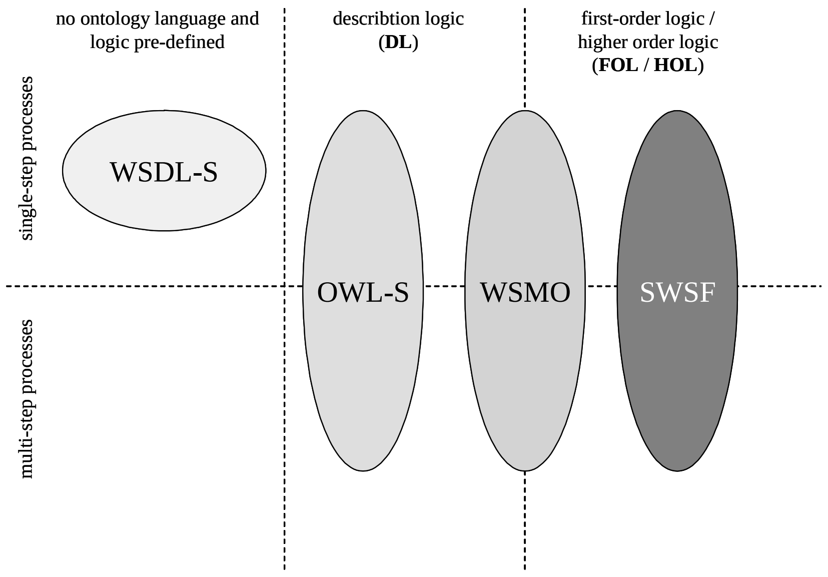

no ontology language and logic pre-defined
describtion logic (DL)
first-order logic / higher order logic (FOL / HOL)
single-step processes
multi-step processes
WSDL-S
OWL-S
WSMO
SWSF

Figure 1: Categorization of Semantic Web Service languages

WSDL-S does not force to use a specific ontology language and therefore it does not define any specific logic. It is only usable for single-step processes whereas the other standards can handle multi-step processes, too. OWL-S builds on OWL which is based on description logic (OWL DL). SWSF uses first-order logic and rules languages to model the ontology in its underlying language SWSL and WSMO offers both logic layers for the modeling of ontologies in WSML.

2.1 UML 2.0

The Unified Modeling Language (UML) is a standard modeling language for visualizing (using the standardized graphic UML notations) and specifying the static structure, dynamic behavior and model organization. UML consists of a notation for de-

scribing the syntax of the modeling language and a graphical notation and a meta-model which describes the static semantics of UML. The UML specification consists of the Infrastructure [1]) which defines foundational language constructs required for UML and of the Superstructure [2] which defines user level constructs (diagrams).
UML in its current version 2.0 offers the modeling of 13 different types of diagrams, six for the modeling of system structures and details of the static system and seven diagrams to model the dynamic behavior of a system.

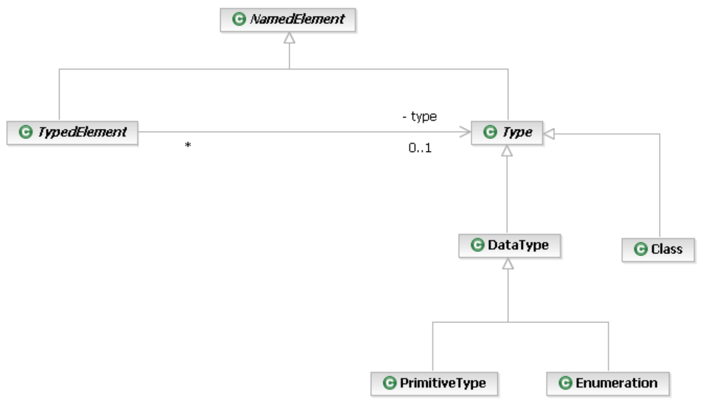


Figure 2: UML Core: type system

The UML Core describes elements that are needed in most of the other packages defined by the UML-Superstructure. An important feature is to have elements with a type, which are called TypedElements in UML and where type can either be a DataType or e.g. a Class (especially important in Class Diagrams for the modeling of software systems).

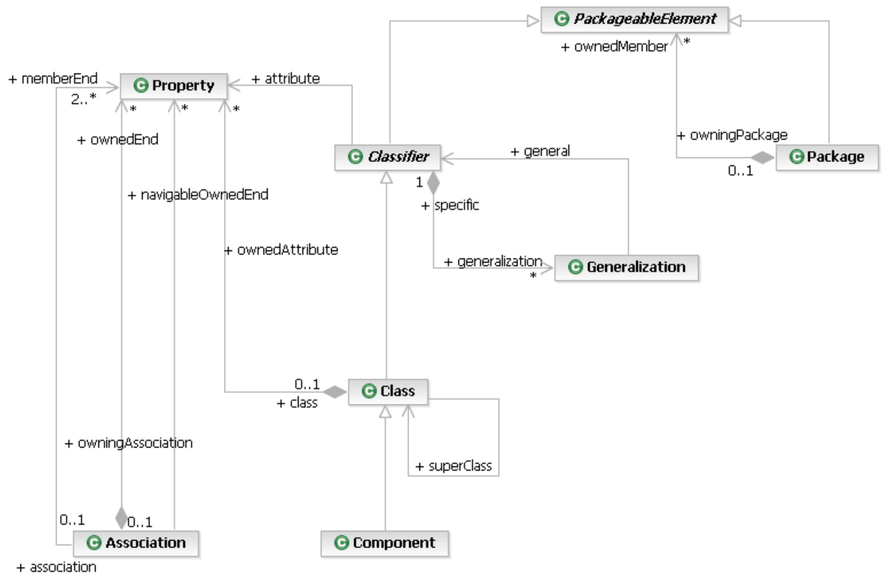


Figure 3: UML Packages, Classes, Properties and Associations

The mostly used UML-elements and graphical notation are defined in the Classes package. Class diagrams are widely common in software methodologies and are used in the analysis (e.g. conceptual modeling of the domain) and design phase (platform independent description as well as platform specific description) to describe classes and interfaces with their attributes, operations and associations (including aggregation and composition), but also generalization and dependencies among them. Class diagrams can be used for the definition of organizational models; in particular the static aspects of the organizations, its associations and part-of relationships can be modeled.

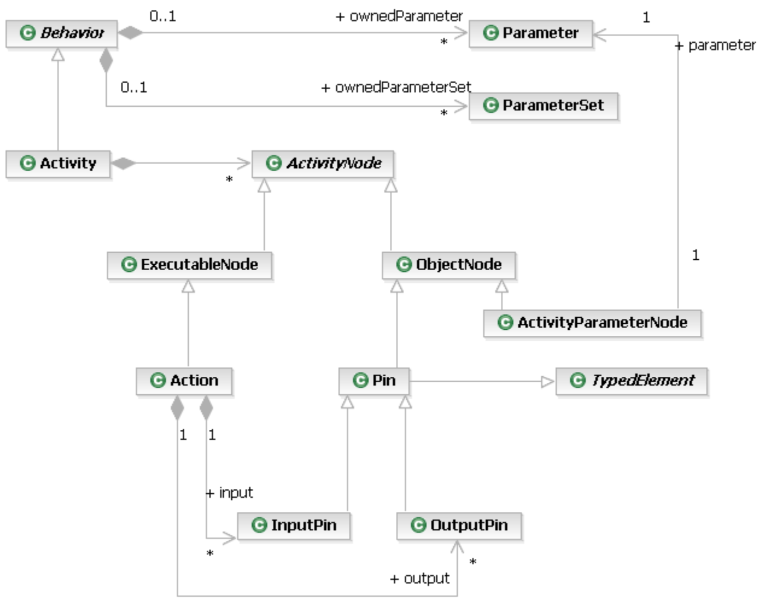


Figure 4: UML Activity: Activity nodes

In the Activity-packages of UML the basic concepts for modeling a process flow are defined. Activity modeling emphasizes the sequence and conditions for coordinating lower-level behaviors. The actions coordinated by activity models can be initiated because other actions finish executing, because objects and data become available or because events occur external to the flow. Each action can have inputs and outputs similar to the whole activity which can have parameters which can be grouped by parameter sets.

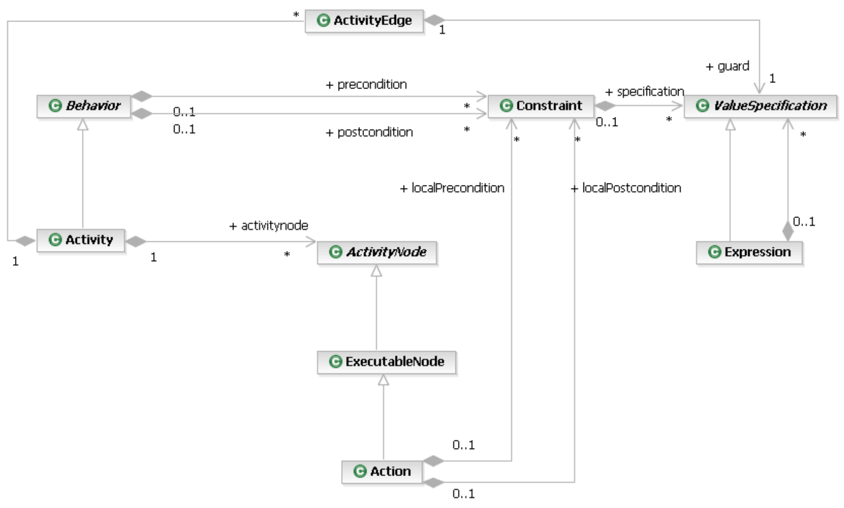


Figure 5: UML Activity: Constraints

One can specify constraints, such as preconditions and effects, on an action as well as on the whole activity. These constraints include expressions in a language such as the Object Constraint Language (OCL).

2.2 OWL-S: Semantic Markup for Web Services

OWL-S (currently version 1.2 is developed) enables the discovery of services (that meet particular requirements and adhere to specified constraints), invocation (by agents or other services), interoperation (through specification of the needed vocabularies), composition (automated service composition and synthesis to provide new services) and verification (of service properties).

OWL-S builds on the formerly developed DAML-S and was one the first submission of a semantic web service to the W3C and currently gets a lot of support in Northern America and Asia.

Each Semantic Web Service in OWL-S consists of a service profile, a service model and a grounding (see Figure 6). The service profile describes what the service does and is used to advertise the service. The service model answers the question “how is it used?” and describes how the service works internally. Finally, the service grounding specifies how to access the service (“how does one interact with it?”).

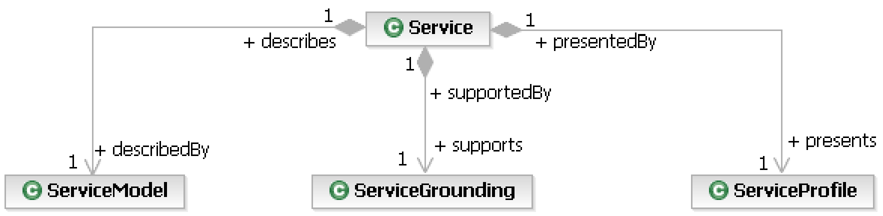


Figure 6: OWL-S meta-model: Service

To give a detailed perspective on how to interact with a service, it can be viewed as a process. OWL-S distinguishes three kinds of processes: Atomic processes, Simple processes and Composite Processes. *Atomic processes* are directly callable and correspond to the actions a service can perform by engaging it in a single interaction; *composite processes* correspond to actions that require multi-step protocols and/or multiple server actions; finally, *simple processes* are not callable and not associated with a grounding and only provide an abstraction mechanism to enable multiple views of the same process.

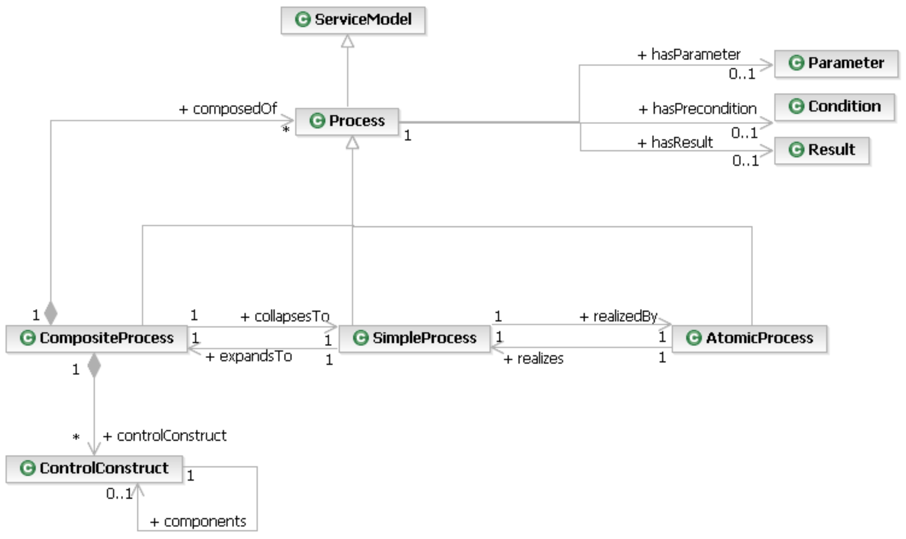


Figure 7: OWL-S meta-model: Service Model

The service profile describes the functional and non-functional (e.g. service category) parts of the process and has several parameters (such as Input or Output) and Expressions like Precondition and Effect.

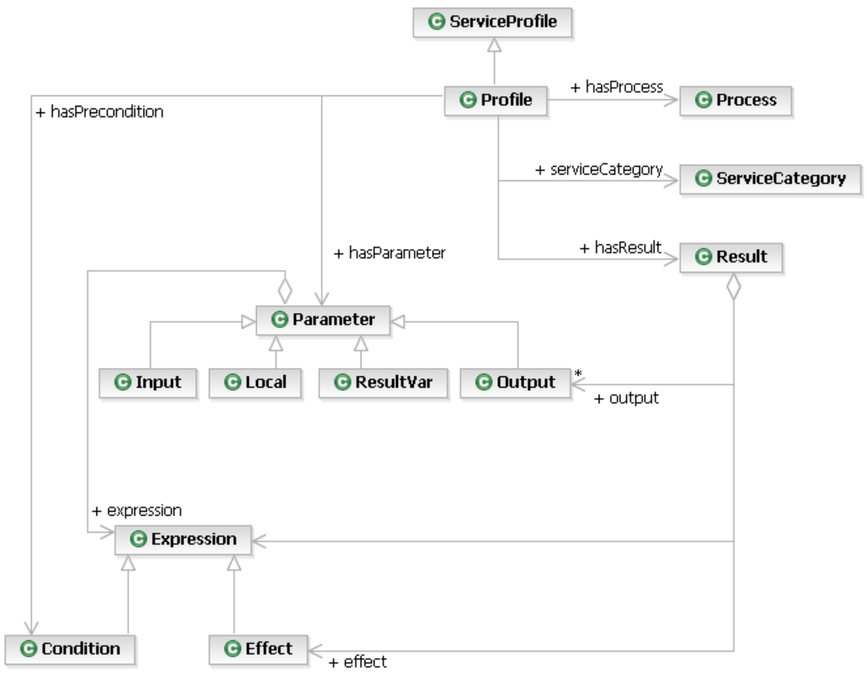


Figure 8: OWL-S meta-model: Service Profile

The grounding of a service specifies the details of how to access the service – details having mainly to do with protocol and message formats, serialization, transport and addressing. Only the service grounding deals with the concrete level of specification, the service profile and model are only thought of as abstract representations. The main aim of grounding is to concretely realize the inputs and outputs of an atomic process as messages. These further carry the inputs and outputs in a specific defined communicable form.

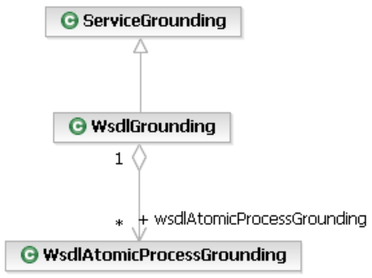


Figure 9: OWL-S meta-model: Service Grounding

OWL-S is based on the Web Ontology Language OWL and supplies web service providers with a core set of markup language constructs for describing the properties and capabilities of their web services in an unambiguous, computer-interpretable form. For a detailed overview of the OWL-S meta-model, see appendix A.

2.3 WSMO: Web Service Modeling Ontology

Based on the Web Service Modeling Framework (WSMF) [26], the Web Service Modeling Ontology Project (WSMO) [7] developed mainly by the Digital Enterprise Research Institute (DERI) is a formal ontology and language that consists of four different main elements for describing semantic web services:

- *Ontologies*: They provide the terminology used by other elements to describe the relevant aspects of the domains of discourse.
- *Goals*: They state the intentions that should be solved by web services and are representations of one or more objectives which need to be fulfilled.
- *Web Services*: A Web Service is a computational entity which is able to achieve a part of or the complete goals a user seeks to fulfill. WSMO web service descriptions describe various aspects of a service and consist of functional, non-functional and the behavioral aspects of a web service.
- *Mediators* to resolve interoperability problems. They describe elements to overcome interoperability and incompatibility problems between different elements on data (ooMediator), process (ggMediator, wgMediator) and protocol level (wwMediator).

WSMO comes along with a modeling language (WSML) [27] and a reference implementation (WSMX). WSML has four variants: WSML-Core which is based on the intersection of Description Logic and Logic Programming and which is the least expressive variant; WSML-DL which is an extension of WSML-Core and offers similar expressive power as OWL-DL and is based on description logic, too. WSML-Flight extends the Core (disjoint to WSML-DL) in the direction of Logic Programming; WSML-Rule which extends WSML-Flight and thus offers the same kind of conceptual modeling features and allows the use of function symbols and unsafe rules. Finally WSML-Full is a superset of WSML-Rule and WSML-DL and can be seen as a notational variant of First-Order Logic with non-monotonic extensions. All WSML variants are specified in terms of a human-readable syntax with keywords similar to the elements of the WSMO conceptual model. WSMO uses F-Logic syntax to provide axioms and logical inference support. F-Logic was originally developed for defining, querying and manipulating database schema. F-Logic combines the advantages of typical frame based languages with the expressiveness, the compact syntax and the well defined semantics from logics.

The EU-funded project WSMO also develops an execution environment (WSMX) which enables discovery, selection, mediation, invocation and interoperation of semantic web services. Besides the reference implementation WSMX, there is the Internet Reasoning Service, IRS-III, which provides the representational and reasoning mechanisms for implementing the WSMO meta-model described below. It supports the developer at design time by providing a set of tools for defining, editing and managing a library of semantic descriptions and also for grounding the descriptions to a standard web service (e.g. described in the Web Service Description Language WSDL).

Most of the elements in WSMO can be described with non-functional properties in greater detail. This includes Quality-of-Service aspects, economic aspects and additional descriptions.

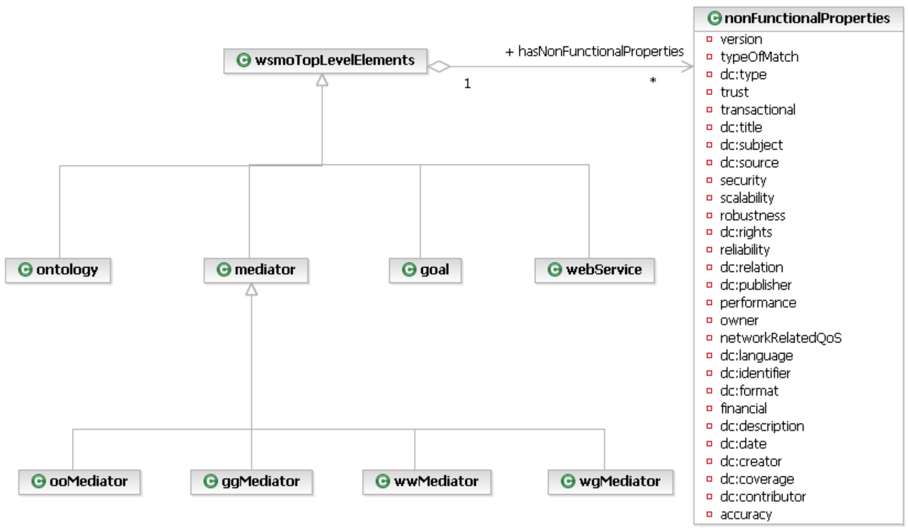


Figure 10: WSMO top level elements

The ontology in WSMO consists mostly of concepts, attributes and instances. Each of them can be described using non-functional properties. Concepts can build a taxonomy and are linked to other concepts using attributes. There can be functions and relations using these concepts as parameter and axioms which are logical expressions together with their non-functional properties.

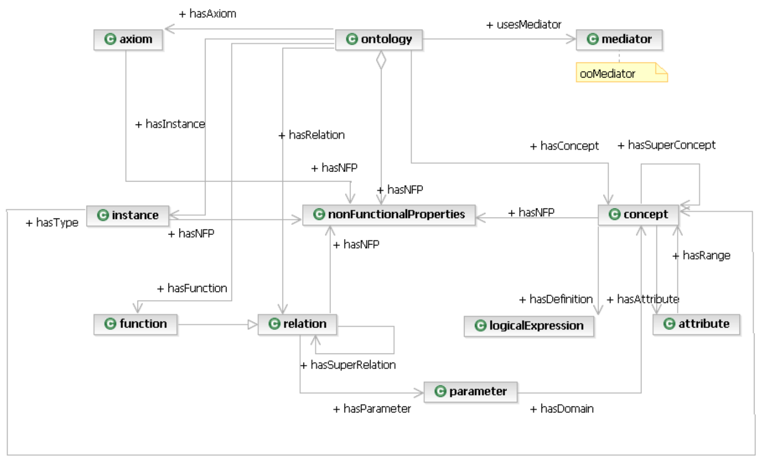


Figure 11: WSMO ontology

Each web service in WSMO builds on one or more ontologies and might use mediators to translate between several ontologies or web services. They are described using interfaces and capabilities. The interface for a complex web service is characterized using the web service orchestration or choreography which is built using an abstract state machine (on states and transitions). The capability describes the functional parameters of the (one-step or multi-step) web service. It specifies the preconditions (the information state of the web service before execution), assumptions (the state of the world before execution), postconditions (the information state of the web service after its execution) and effects (the state of the world after execution).

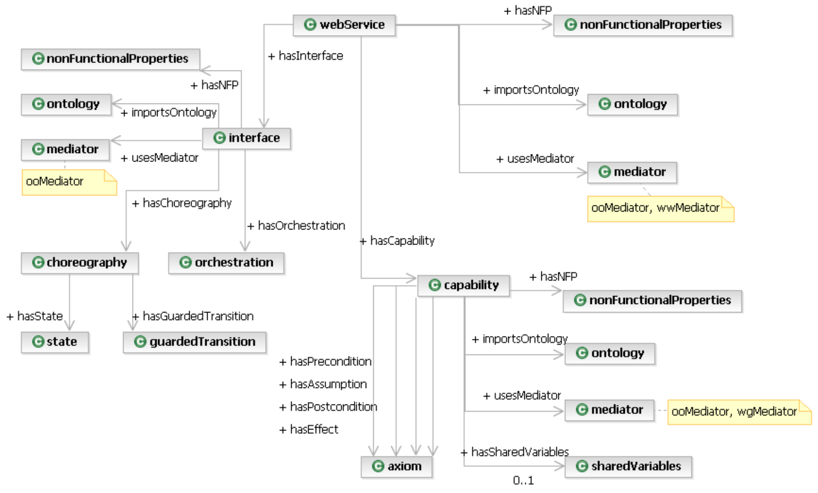


Figure 12: WSMO web service

WSMO goals enable the possibility to model the intentions that should be solved by one or more web services. They describe an interface that should be implemented and capabilities that need to be fulfilled. Therefore, ontologies and mediators might be needed to achieve the goal.

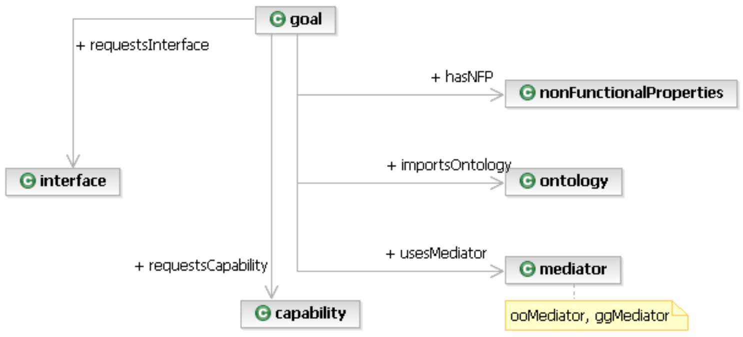


Figure 13: WSMO goals

2.4 WSDL-S: Web Service Semantics

WSDL-S is another semantic web service approach submitted to the W3C in November 2005 and extends the WSDL standard with semantic information (using a new attribute modelReference). Currently it only considers single-step services and there is no specification how a composition of several web services should be handled. However, there are already efforts to annotate web service choreography languages like WS-BPEL similar to the annotations made in WSDL-S (see e.g. [29]). WSDL-S uses the extensibility elements of WSDL and introduces new elements to describe the semantics of a service, of its inputs and outputs and of preconditions and effects.

All input and output types can be annotated with semantic elements and additional "mediators" (as they have been called in WSMO) can be defined using XSLT-transformations. Each operation can have a semantic annotation and for each interface categories can be defined (similar to OWL-S).

The advantages of this approach can be summarized as follows:

- it builds on existing web services standards
- it supports the user's choice of the semantic representation language
- it allows the association of multiple annotations
- it supports semantic annotation of web services whose data types are described in XML schema (therefore e.g. GRDDL [10] could be used)

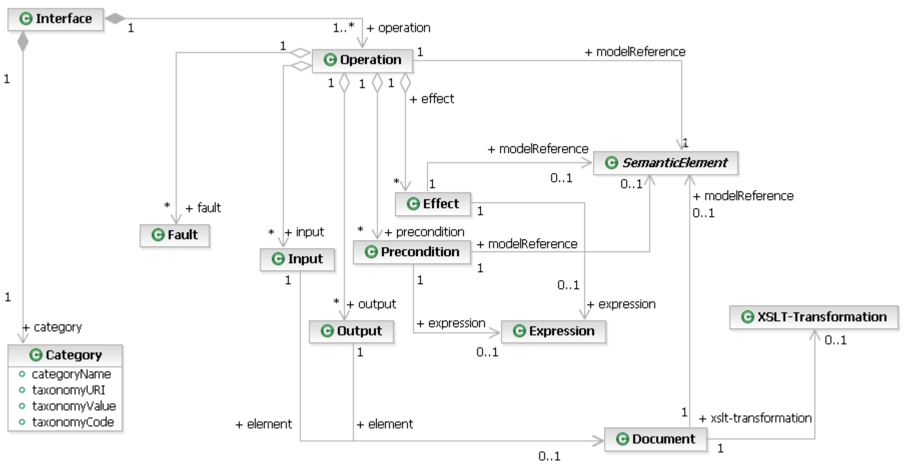


Figure 14: WSDL-S meta-model

2.5 SWSF: Semantic Web Services Framework

Another W3C submission (from September 2005) beside OWL-S and WSMO is the Semantic Web Services Framework (SWSF) [9] which represents an attempt to extend the work of OWL-S and consists of two major parts:

- the language ***SWSL*** as an underlying basis: The Semantic Web Service Language has two sublanguages:
    - *SWSL-FOL* which is based on first-order logic and primarily used to express the formal characterization of web service concepts and
    - *SWSL-Rules* that is similarly to WSML-Rule based on logic programming; and
- the ontology ***SWSO*** above. The Semantic Web Service Ontology presents a conceptual model by which web services can be described and, again, which can be divided in two forms:
    - *FLOWS*, the first-order logic ontology for web services which has (following the high-level structure of OWL-S) three major components: Service Descriptors, Process Model and Grounding. It adapts the key concepts from the Process Specification Language PSL (ISO 18629) which was originally developed to enable sharing of descriptions of manufacturing processes. A fundamental building block of FLOWS is the concept of an atomic process (similar to OWL-S). Associated with an atomic process are zero or more parameters that capture the inputs, outputs, preconditions and effects (simply called ***IOPE***s)
    - *ROWS*, the rules ontology for web services which enables implementations in reasoning and execution environments based on logic-programming.

SWSF emerged from the work in service composition which might require more expressivity than is available in OWL and is therefore based on logic programming, first-order logic and policy research. It builds on DAML-S, OWL-S and WSMO and provides rich semantics for greater automation of discovery, selection and invocation, content transformation, composition, monitoring and recovery and verification. Lying the focus on the messages similar to WSDL 2.0 (where several Message Exchange Patterns (MEP) have been developed), it introduces the concepts of Channels and Messages which can be created and modified using several specialized actions.

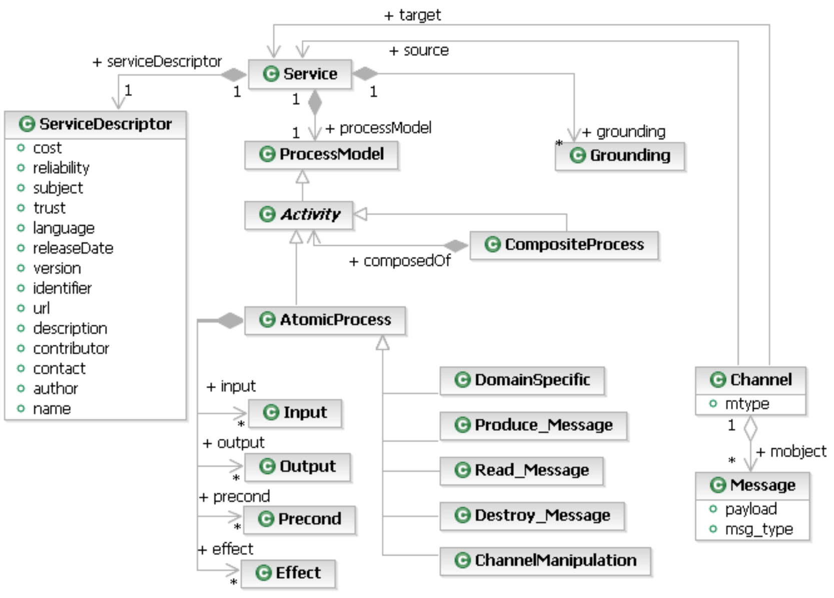


Figure 15: SWSF Meta-model: FLOWS Core

To compose several services in a multi-step process, it defines control constraints (in the style of OWL-S) for alternatives, loops, parallel flows, choices, etc. Additionally, it defines ordering constraints to allow the specification of activities defined by sequencing properties of atomic processes (OrderedActivity), Occurrence Constraints to support the specification of nondeterministic activities within services (OccActivity), State Constraints (for activities which are triggered by states) and Exception Constraints.

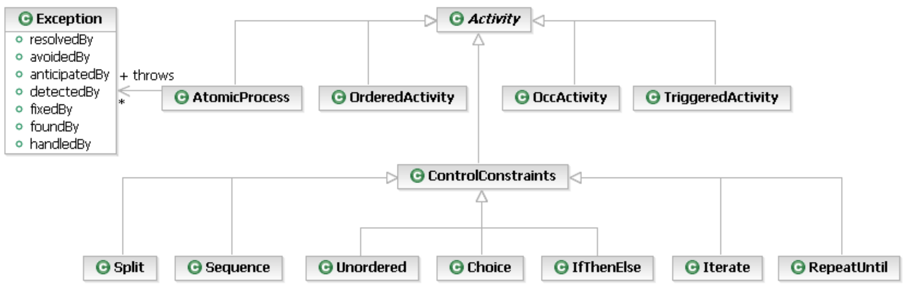


Figure 16: SWSF meta-model: Other Constraints

2.6 Other Semantic Web Services approaches

Another approach for enhancing web services with semantic information is the Web Services Description Language Version 2.0: RDF Mapping (Working draft, 18 May 2006) [11] which defines mapping rules how the constructs from WSDL could be transformed into an ontology using constructs defined in RDF and OWL (or other languages). Another attempt is the W3C working draft "Semantic Annotations for WSDL (SAWSDL)" [30] which introduces new WSDL elements "modelReference", "liftingSchemaMapping" and "loweringSchemaMapping" based on the extensibility elements of WSDL 2.0. Because these approaches are comparable with WSDL-S and are still work in progress, we will not go into further detail or consider them in our meta-model.

SESMA [25] is another approach for a semantic web service description format which allows a tight integration with existing web service standards like WSDL, SOAP and WS-BPEL. It provides a precisely defined semantics and modeling constructs as logical variables and formulas. It claims to be truly complementary to existing standards and extensible for future requirements. Hence, it is neither widely distributed nor submitted to the W3C which is the reason why we will not consider this approach in our meta-model.

2.7 ODM: Ontology Definition Metamodel

Based on the ideas and work of Stephen Cranefield, Dragan Gašević and others, the Object Management Group (OMG) works on a specification for the interoperability between ontologies (and its underlying description logic), UML diagrams, ER-diagrams, Topic Maps and common logic. The Ontology Definition Metamodel (ODM) [5] consists of four platform independent models (PIM: Common logic, Topic Maps, RDF and OWL) and informative models like the one for Description logic. It defines a meta-model and UML-profile for RDF(S) and OWL-ontologies and mappings between the meta-model and these semantic web standards.

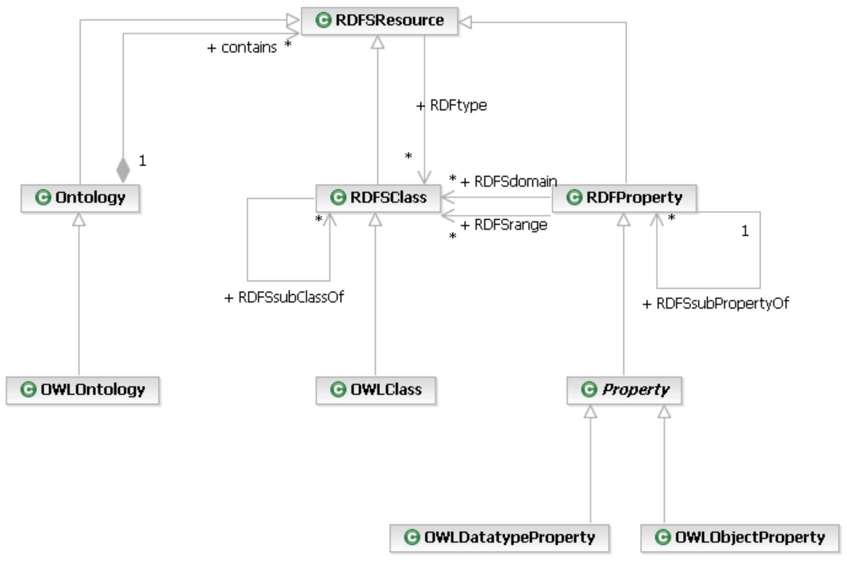


Figure 17: ODM: ontology, classes and properties

Each concept in an ontology is a RDFSResource (including the ontology itself). There are classes and properties which can be connected and enable an easy modeling of ontologies (see Figure 17 for a very short excerpt of the ODM draft).

3 A meta-model for semantic web services

Based on the above described standards and having the need for an independent way of modeling semantic web services in mind, we designed a meta-model which can be applied and transformed into all of the mentioned W3C submissions.

3.1 Overview

Our meta-model builds on existing semantic web service approaches, on the standard ODM for the modeling of ontologies and, of course, on UML. Each semantic web service standard builds on different languages: OWL-S on RDF and OWL, WSMO on WSML, SWSF on SWSL. Only WSDL-S stays independent on any ontology language.

Our semantic web service meta-model is based on the Ontology Definition Meta-model for modeling ontologies (in form of a UML-profile) and on the above mentioned semantic web service submissions. WSDL-S is independent on the underlying ontology modeling language, but OWL-S, WSMO and SWSF require a specific ontology language (OWL, WSML and SWSL).

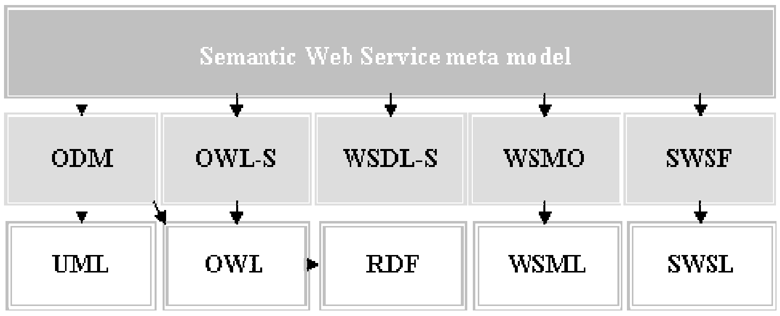


Figure 18: Profile dependencies

It is difficult to integrate our meta-model into the layers of MDA. It is platform independent in principal, but also includes constructs for each specific platform and semantic web service language and one can directly generate code from the meta-model. But, this matches to our own experiences that business users prefer one meta-model avoiding model transformations where possible and use different views on a meta-model instead. This also conforms to current discussions about the future of the MDA at the OMG where e.g. Stan Hendryx, Chairman of the OMG Business Rules Special Interest Group, pleaded for a weakening of the current layers.

3.2 Packages

Our profile consists of five packages which interact with each other. The Ontology-package contains all concepts that are needed to model an ontology (similar to ODM). The Interfaces-package provides all elements to model a WSDL service and to describe it with semantics (like in WSDL-S). The ServiceProvider-package includes all aspects to model one or more semantic web services with non-functional descriptions and using the elements of the ProcessFlow-package the functional elements and composition of multi-steps can be modeled. Every single step can be annotated with functional descriptions as described in the Functional-package. The Interfaces-package, the ProcessFlow-package and the Functional-package access elements of the ontology and therefore import the Ontology-package. The ProcessFlow-package extends concepts defined in the ServiceProvider-package and therefore merges this package. The Functional-packages extends some concepts of the ProcessFlow-package and therefore merges this, too.

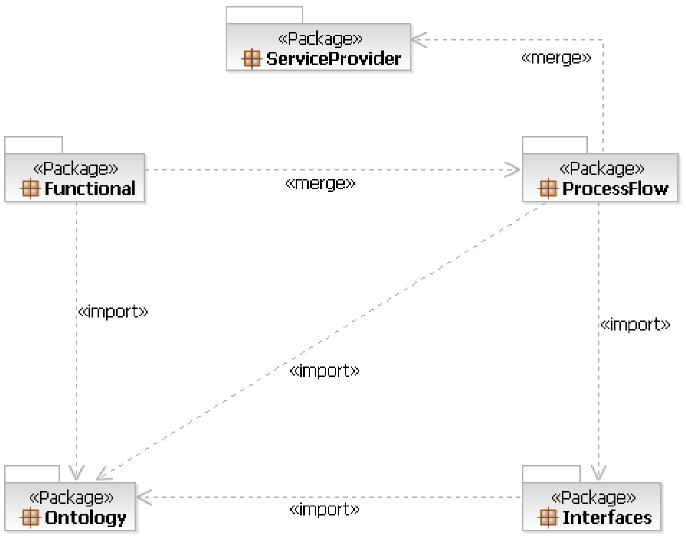


Figure 19: Package overview for our meta-model

At the bottom of our meta-model are elements to model the constructs in an ontology (whereby it doesn't matter whether this ontology needs to be in OWL or in WSML). To model all kind of syntactic and semantic data, the top-level element of our ontology-package (compare Figure 20) is a *DataElement*. A *DataElement* can be a *SyntacticElement* (e.g. an element specified in an XSD-file: *XSDType*) or it can be a *SemanticElement*. This might be (referring to ODM) a *RDFSResource* which is part of an ontology. A resource can either be a class (or concept as it is called in WSML) or a property. *RDFSClass*es can be connected to other classed via *RDFProperties* which contain the domain and range of the property. Every class (resp. property) can be generalized and there exist specializations for OWL classes which are the mostly used presentation form of current ontologies.

The *DataElement* and *XSDType* are necessities from OWL-S where it should be possible to reference not only semantic elements as input or output, but also elements which are not semantically described. The other classes are directly adapted from ODM, where some of the classes in ODM have been neglected for the sake of simplicity.

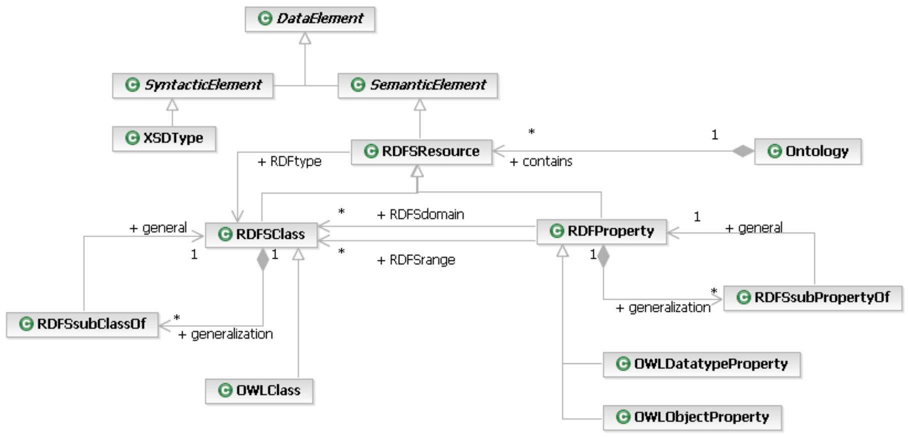


Figure 20: SWS meta-model for the ontology

To model the (syntactical) infrastructure of the services, the interfaces package can be used (Figure 21). Every web service (as defined in WSDL) has an *Interface* which includes a number of operations. An *Operation* is callable from other web services. Every operation can have one input and output *Message* and zero or more fault messages. These messages are exchanged through channels between at least two services. Each operation might be described with a semantic element from the ontology. A message contains one or more *Document*s which can also be described with semantic elements and which can be structured with several *Attribute*s which might themselves be documents again or simple data types (like String, int, etc.). *Interfaces*, *Operations*, *Messages*, *Documents* and *Attributes* are needed to generate WSDL code, the dependencies to the *SemanticElements* have their origin in the WSDL-S approach.

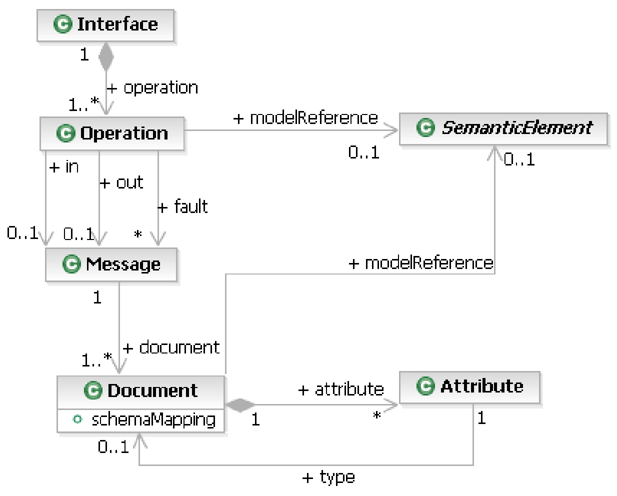


Figure 21: SWS meta-model for interfaces, operations and messages

With these basic constructs for describing the interfaces, operations, messages and the underlying ontology we can now start the modeling of one or more semantic web services. A web service is a *Process* with a specified behavior. A *ServiceProvider* (e.g. an institution, person or organizatrion) offers a number of processes which can be executed. Every process can be categorized (according to OWL-S and SWSF) with a *categoryName*, a path to a *taxonomy*, a specific *value* in the taxonomy and the *code* associated to a taxonomy. Each processs can be additionally described with a *ServiceDescriptor* which includes a serviceName (name), textDescription (description) and contactInformation (contact_info) as needed in OWL-S and additional information which are needed in SWSF such as the author of the service, the version, releasedate, etc. Some more non-functional descriptions are defined in the WSMO-approach which are covered in the properties of the class NonFunctionals. The whole class and properties of *NonFunctionals* can be neglected, if the user can be sure that he/she doesn't want to generate WSMO code. Each process can communicate with other processes via *Channel*s (which is a requirement from SWSF) where the above introduced messages are exchanged.

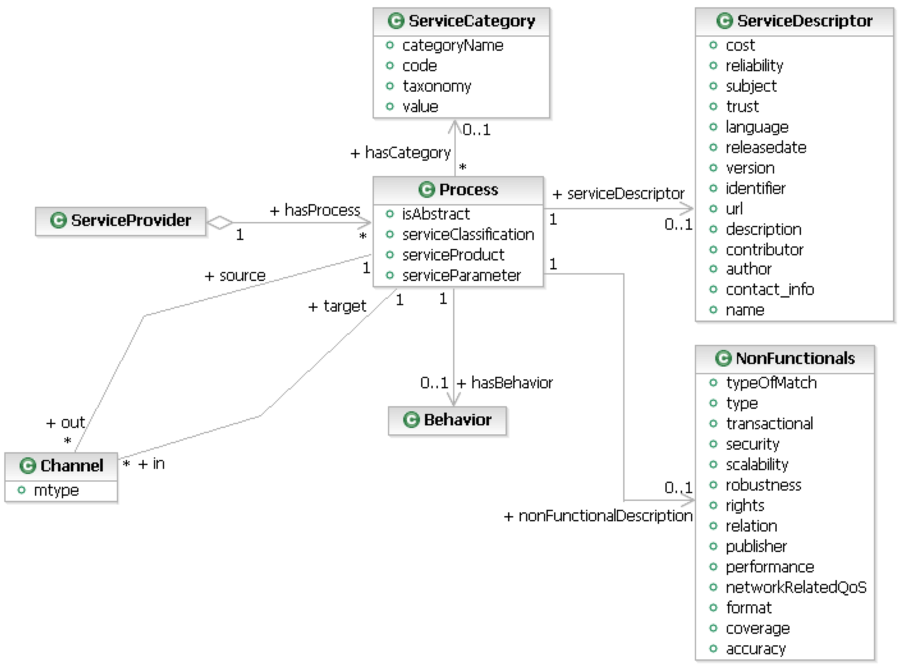


Figure 22: SWS meta-model for a service provider

Each process has an internal behavior which can be described using a *ProcessFlow*. A process flow contains *Node*s and *Connection*s between these nodes. A node might either be a control node like the ones known in UML activity diagrams (InitialNode, ForkNode, JoinNode, DecisionNode, MergeNode, etc.) or an *Action* or a *Composite-Process* which contains several other nodes. A *CompositeProcess* describes a multi-step process whereas a ProcessFlow with only one node specifies a single-step process. An *Action* can be either an *AtomicProcess* which calls an operation or a *Call-CompositeProcess* which can start a new behavior or a *CompositeProcess*.

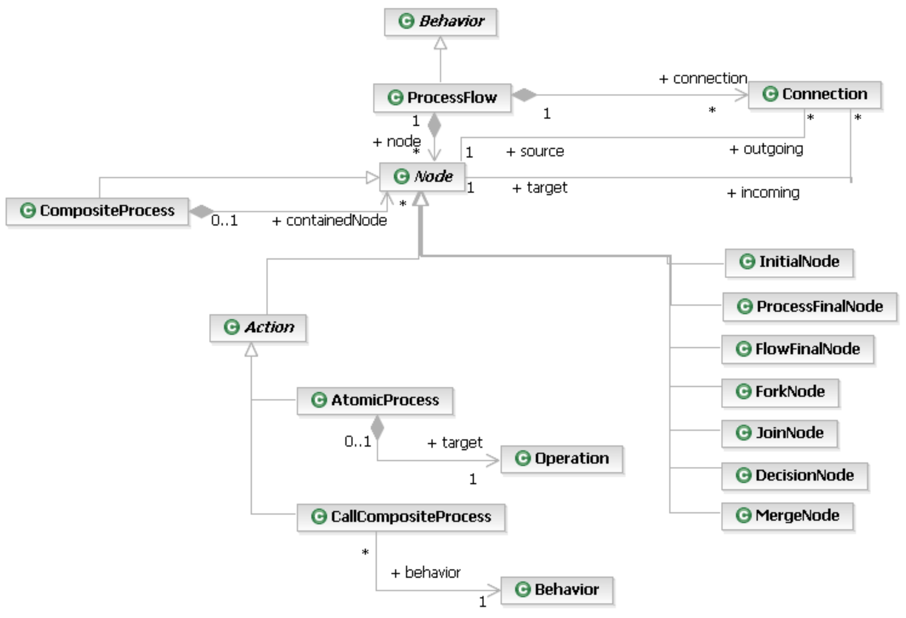


Figure 23: SWS meta-model for the process flow

Each action might require *Input*s and produce *Output*s which can be described more detailed as *DataElements* as introduced above. Every action can also be described with its *Preconditions* and *Effect*s, which itself can be described in more detail with a class or concept of the ontology. The *Preconditions* describe the necessary information state and state of the world before an execution of the action is possible, the *Effects* show how the state of the world and the information state have changed after executing the action.

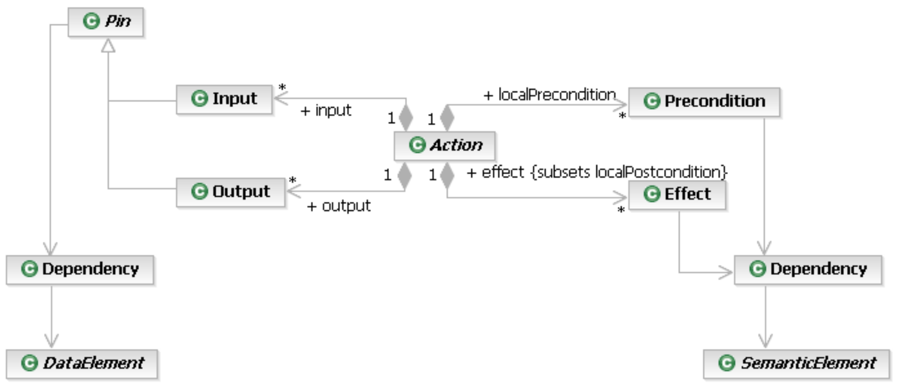


Figure 24: SWS meta-model for the functional description of an action

4 A UML 2.0 profile for our meta-model

To use the meta-model introduced above, we created a UML 2.0 profile and implemented this profile in a UML tool suite. We shortly describe how each element of our meta-model can be stereotyped from classes of the UML specification. Sadly, our modeling tool didn't support filled arrows for the specification of the stereotypes, why we use generalizations in this chapter instead.

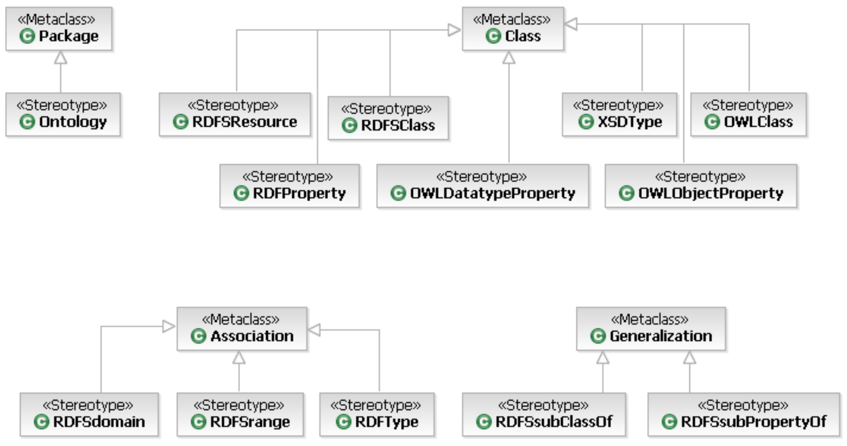


Figure 25: Stereotypes: Ontology

Stereotype	Metaclass	Description
Ontology	Package	An ontology-package describes all elements (classes or concepts and its relations) of the underlying ontology.
RDFSResource	Class	RDFSResource is every element in an ontology.
RDFSClass	Class	RDFSClass is a class or concept in an ontology.
OWLClass	Class	OWLClass is the class of an OWL-ontology.
XSDType	Class	XSDType is an element of an XML Schema document.
RDFProperty	Class	RDFProperty is a property (or relation) between two RDFSClasses.
OWLDatatypeProp-erty	Class	This describes attributes of an OWL class which have a simple datatype.
OWLObjectProperty	Class	Describes attributes of an OWL class which are themselves objects (classes) in the ontology.

RDFType	Association	Specifies the type of a class or concept.
RDFSdomain	Association	Specifies the domain of a property.
RDFSrange	Association	Specifies the range of a property.
RDFSsubClassOf	Generalization	Declares that the first class is subclass of the second.
RDFSsubPropertyOf	Generalization	Declares that the first property is sub-property of the secnd.

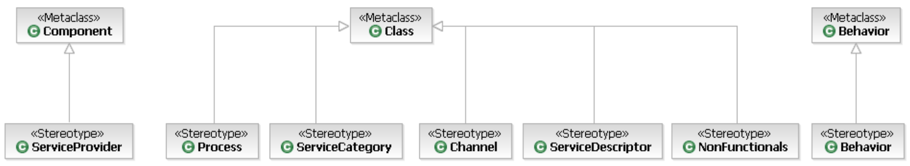


Figure 26: Stereotypes: ServiceProvider

Stereotype	Metaclass	Description
ServiceProvider	Component	A collection of processes that are provided by one institution.
Process	Class	A web service (either a single-step or consisting of multiple steps, optional: abstract or not)
ServiceCategory	Class	The category a web service can be allo-cated.
ServiceDescriptor	Class	Additional information and descriptions about a web service.
NonFunctionals	Class	More non-functional descriptions about a web service.
Channel	Class	The physical connection between two processes.
Behavior	Behavior	The internal behavior a web service has.

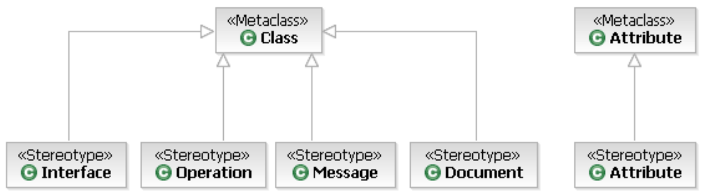


Figure 27: Stereotypes: Interfaces

Stereotype	Metaclass	Description
Interface	Class	A WSDL interface

Operation	Class	The operation where a web service can be called.
Message	Class	Describes the messages that are exchanged in channels between web services.
Document	Class	The content of messages.
Attribute	Attribute	Possibility to structure documents.

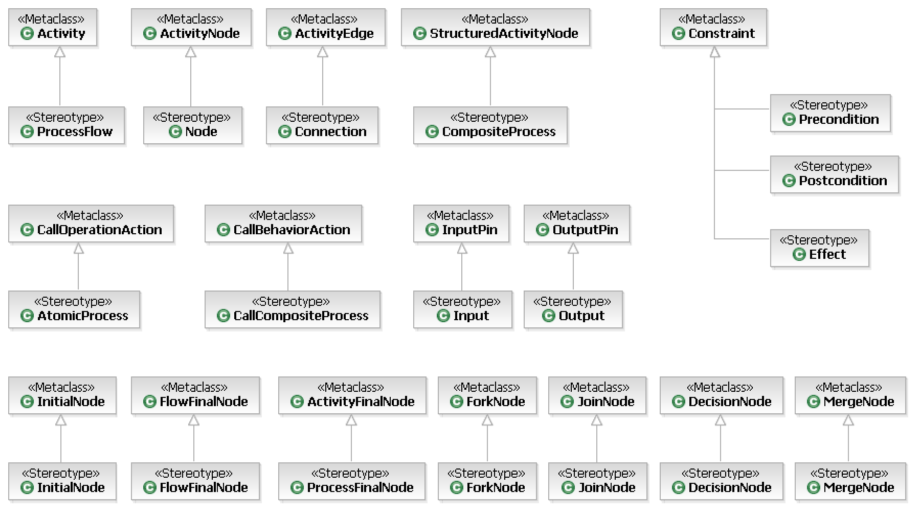


Figure 28: Stereotypes ProcessFlow and Functional

Stereotype	Metaclass	Description
ProcessFlow	Activity	The internal behavior of a process.
Node	ActivityNode	Each step of a web service.
Connection	ActivityEdge	Describes the flow of two or more steps.
CompositeProcess	StructuredActivityNode	An arrangement of more steps which itself is a web service again.
AtomicProcess	CallOperationAction	The invocation of another web service (via a channel and messages).
CallCompositeProcess	CallBehaviorAction	The invocation of another complex (multi-step) web service which has been modeled, too.
Input	InputPin	The inputs each action needs to be able to start.
Output	OutputPin	The data an action generates.
Precondition	Constraint	The information space and state of the world that are necessary to

		start the action.
Effect	Constraint	The state of the world after executing the action.
InitialNode	InitialNode	Start point of the process flow.
FlowFinalNode	FlowFinalNode	End point of the process flow.
ProcessFinalNode	ActivityFinalNode	End point of the process flow.
ForkNode	ForkNode	Start of several parallel threads.
JoinNode	JoinNode	Join of several parallel threads.
DecisionNode	DecisionNode	Start of several alternative threads.
MergeNode	MergeNode	Merging of several alternative threads.

5 Overall example

As an overall example, to show that our profile meets the requirements to generate automatically semantic web service code, we will introduce a well-known example that has been created within the OWL-S specification. The example is called CongoBuy and is a B2C bookbuying example showing the OWL-S usage, illustrating a simple use of the process model. The service described is a fictional book buying (or selling, depending on your perspective) service from www.congo.com. The example is divided in two parts: ExpressCongoBuy as a very small example with only one web service invocation (single-step) and FullCongoBuy as a multi-step web service composition. Where appropriate we will model the ExpressCongoBuy to show the concepts of our profile. In parts where the small example can't explain concepts of our profile, we switch to the FullCongoBuy-example.
First, the ontology for this example needs to be modelled (see Figure 29) to cover the most important concepts that are needed in this example.

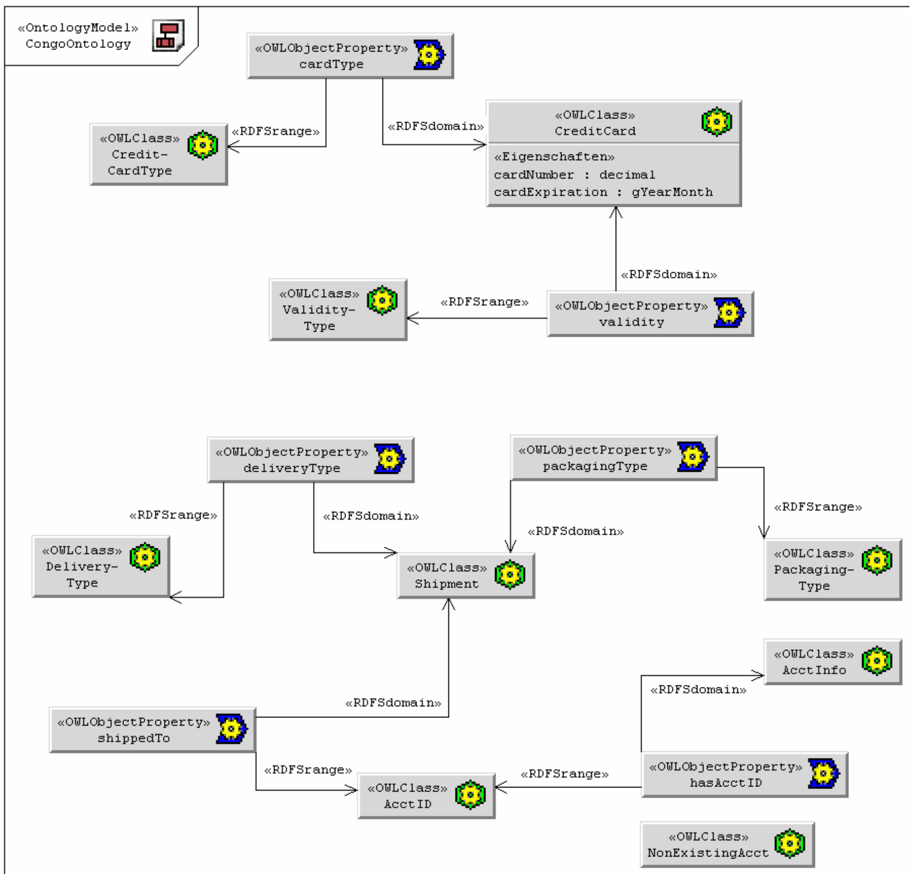

«OntologyModel» CongoOntology
«OWLObjectProperty» cardType
«RDFSrange»
«RDFSdomain»
«OWLClass» Credit-CardType
«OWLClass» CreditCard
«Eigenschaften»
cardNumber : decimal
cardExpiration : gYearMonth
«RDFSdomain»
«OWLClass» Validity-Type
«RDFSrange»
«OWLObjectProperty» validity
«OWLObjectProperty» deliveryType
«OWLObjectProperty» packagingType
«RDFSrange»
«RDFSdomain»
«RDFSdomain»
«RDFSrange»
«OWLClass» Delivery-Type
«OWLClass» Shipment
«OWLClass» Packaging-Type
«OWLClass» AcctInfo
«RDFSdomain»
«RDFSdomain»
«OWLObjectProperty» shippedTo
«RDFSrange»
«OWLClass» AcctID
«RDFSrange»
«OWLObjectProperty» hasAcctID
«OWLClass» NonExistingAcct

Figure 29: Example: CongoOntology

Afterwards, one needs to create all documents which are needed to exchange data between processes. In the CongoBuy-example no documents have been specified, therefore we only show the usage of two simple documents which have been nested.

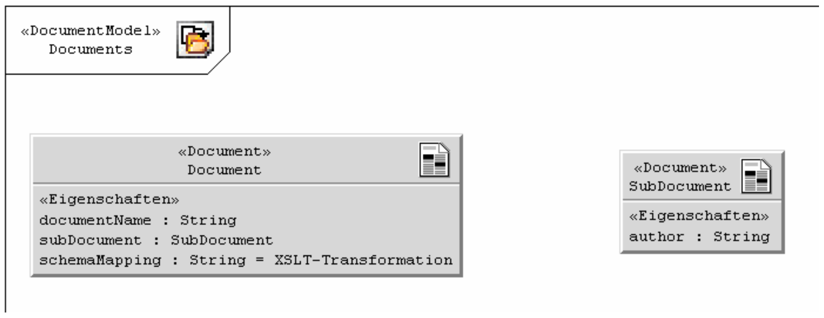


Figure 30: Example: Documents

The messages which are exchanged between two services need to be modeled next. The ExpressCongoBuy-example has two inputs and three outputs (creditCardNumber, creditCardType and creditCardExpirationDate).

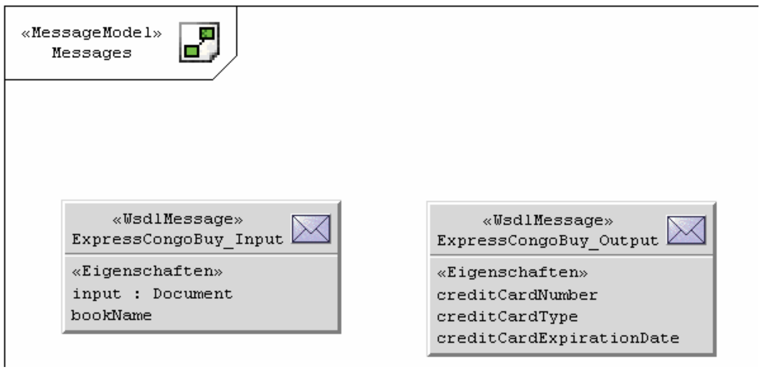


Figure 31: Example: Messages

The above modeled messages then can be used to model the interfaces and its operations. The ExpressCongoBuy-example only has one interface, the FullCongoBuy-example has much more operations which were not modeled in fully detail here.

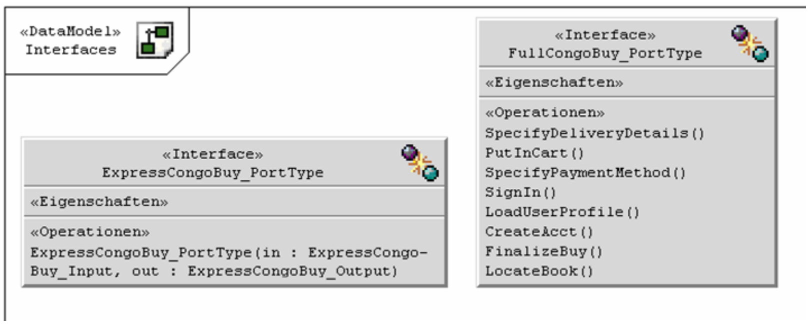


Figure 32: Example: Interfaces

Additional data types which are needed to model the inputs and outputs of the process are modeled in an own diagram. These elements can be imported from existing XSD-files or modeled from stretch.

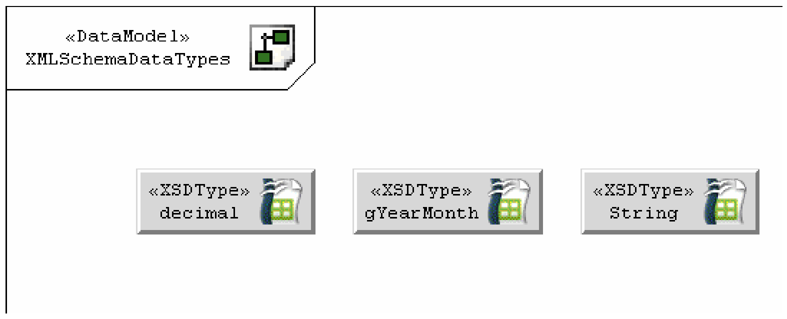


Figure 33: Example: XSD-Types

The next diagram specifies the details about each web service (Process). A service provider (here: Congo) can have multiple processes (ExpressCongoBuy and Full-CongoBuy) which itself can be described with additional information. They can be categorizes and described with functional and non-functional information (e.g. cost or trust). Similar to the attribute “isAbstract” which has a concrete type and a standard value, each of the attributes can be given a value which has been suppressed here for the sake of simplicity.

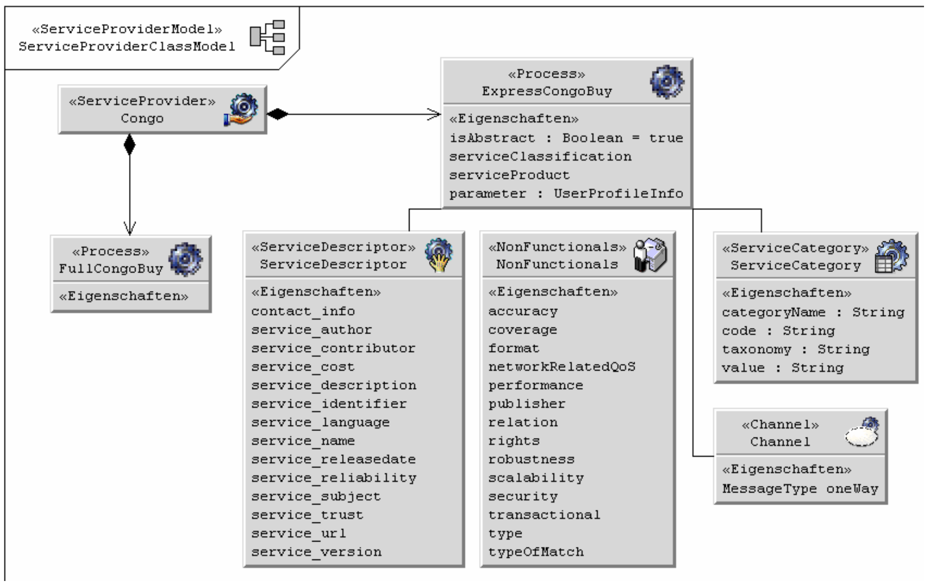


Figure 34: Example: ServiceProvider with processes

Each process has a behavior and in the example of ExpressCongoBuy this behavior consists of one simple action. This action has several inputs and one output and it can be described in more detail with preconditions and effects. These can be specified in OCL, KIF or SWRL, where the former is a language only used in UML diagrams which needs to be transformed into a semantic web language and the latter two are used in several semantic web service languages.

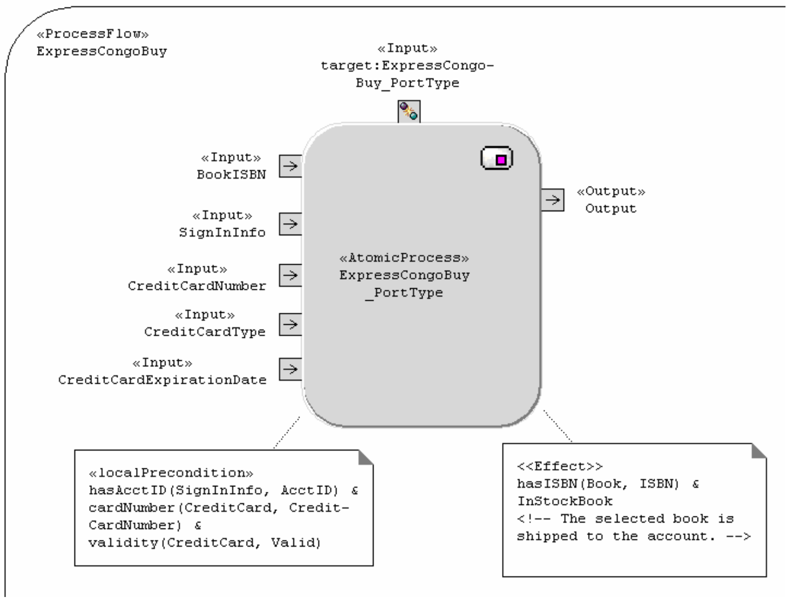


Figure 35: Example: ExpressCongoBuy atomic process

The FullCongoBuy-example does not only consist of one action, but of multiple actions (multi-step or CompositeProcess) which are coordinated using the control nodes introduced above. These actions can be combined to composite processes themselves again. To keep it simple we removed the more detailed description of each action (meaning inputs, outputs, preconditions and effects) in Figure 36.

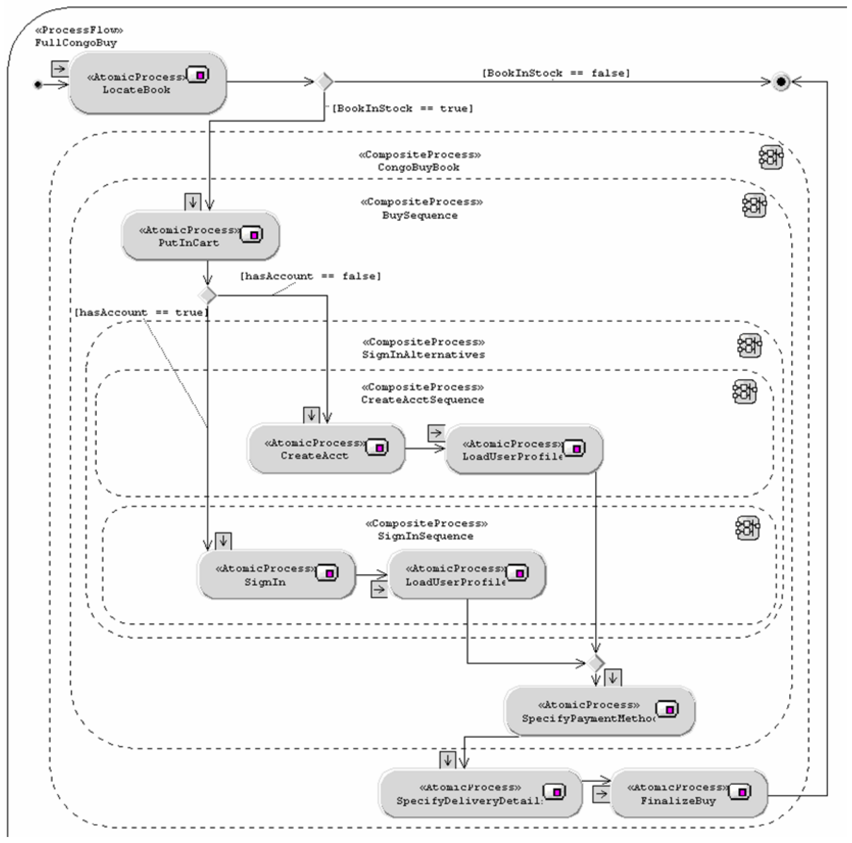

«ProcessFlow» FullCongoBuy
«AtomicProcess» LocateBook
[BookInStock == false]
[BookInStock == true]
«CompositeProcess» CongoBuyBook
«CompositeProcess» BuySequence
«AtomicProcess» PutInCart
[hasAccount == false]
[hasAccount == true]
«CompositeProcess» SignInAlternatives
«CompositeProcess» CreateAcctSequence
«AtomicProcess» CreateAcct
«AtomicProcess» LoadUserProfile
«CompositeProcess» SignInSequence
«AtomicProcess» SignIn
«AtomicProcess» LoadUserProfile
«AtomicProcess» SpecifyPaymentMethod
«AtomicProcess» SpecifyDeliveryDetails
«AtomicProcess» FinalizeBuy

Figure 36: Example: FullCongoBuy composite process flow

6 Transformation rules

The following sections describe the transformation from our meta-model and UML-profile to each semantic web service standard. Starting with OWL-S which includes the generation of the ontology in OWL, the WSDL code for the interface and the semantic description in OWL-S, WSMO, WSDL-S and SWSF-code will be generated. The transformation rules are described in an informal way, only to show the coherences between our meta-model and each SWS-language. For detailed transformation and code generation rules, please have a look at chapter 7.

6.1 Transformation to OWL-S

To generate OWL-S code the ontology in RDF and OWL needs to be generated first. After that the WSDL-file including interfaces, operations, messages, etc. is produced, before the semantic descriptions that are subject of the OWL-S submission are considered.

6.1.1 Generation of the ontology

The generation of the ontology in OWL from our meta-model confirms to the specification of the Ontology Definition Metamodel ODM. Each OWLClass gets a class in the ontology, each OWLDatatypeProperty a DatatypeProperty in OWL and every OWLObjectProperty an ObjectProperty in OWL. For a more detailed description, refer to [5].

6.1.2 Generation of WSDL code

The web service description itself is an interface with several operations that a web service offers. Each operation contains a maximum of one input and output message and zero or more fault messages.

A WSDL document consists of a number of definitions. The root of every WSDL document is the element *definitions*, which contains all other definitions.

The rool element is created at the beginning of the transformation.

```
<wsdl:definitions name=„Definition.Name“
        targetNamespace=„http://www.mid.de/example/“>
        ….
</wsdl: definitions>
```

Data definition and messages:

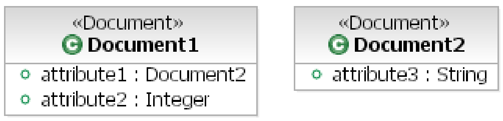
A message is an abstract description of exchanged data and consists of one or more logical parts. Every part of a message is connected to a datatype. Parts offer the flexible possibility to describe message contents in an abstract way. Classes with the stereotype «Document» in our meta-model are transformed to elements from a type definition in WSDL. For each «Document» an element and a complex type is generated in WSDL. The name of the element and of the complex type is the name of the Document. The type-attribute references the complex type. The «Attribute» of a document gets an element of the complex type in WSDL. The name of the element is the name of the attribute. The datatyp might either be another complex type or a standard type like String or int.

```
<wsdl: types>
  <xsd:schema>
    <complexType name=„Document.Name“>
      <all>
        <element name=„Attribute.Name“ type=„Attribute.Type“/>
        …
      </all>
    </complexType>
    <element name=„Document.Name“ type=„tns:Dokument.Name“/>
      …
  </xsd: schema>
</wsdl: types>
```

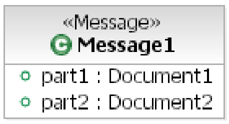
For each class with the stereotype «Message» a new message element in WSDL gets created. A part of the message is pasted for each attribute of a message.

```
<wsdl:message name=„Message.Name“>
   <wsdl:part name=„Attribute.Name“ element=„tns:Document.Name“/>
   ...
</wsdl:message>
```

For example:



```
<wsdl: types>
  <xsd:schema>
    <complexType name=„Document2“>
        <all>
          <element name=„attribute3“ type=„string“/>
        </all>
    </complexType>
    <element name=„Document2“ type=„tns:Document2“/>
    <complexType name=„Document1“>
        <all>
          <element name=„attribute1“ type=„tns:Document2“/>
          <element name=„attribute2“ type=„int“/>
        </all>
    </complexType>
    <element name=„Document1“ type=„tns:Document1“/>
  </xsd: schema>
</wsdl: types>
```



```
<wsdl:message name=„Message1“>
        <wsdl:part name=„part1“ element=„tns:Document1“/>
        <wsdl:part name=„part2“ element=„tns:Document2“/>
</wsdl:message>
```

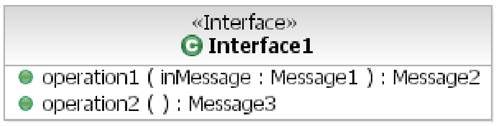
Interfaces and operations:

An operation is a abstract description of an action which can be executed from a service. It receives input-messages and gives output-messages back to the requester. Optionally when an error occurs it can send one or more fault-messages.
For each class from the stereotype «Interface» a WSDL portType is generated. The name of the portType is the same as the name of the interface. For each operation of an interface a new operation element in WSDL is created as part of the portType. For each message (input, output or fault) a new definition as part of the WSDL-operation is created. These definitions refer with the attribute message to a unique message definition. There are several operation types for the definition of the message order:

- one-way: input
- request-response: input – ouput – fault*
- solicit-response: output – input – fault*
- notification: output

```
<wsdl:portType name =„Interface.Name“>
        <wsdl:operation name=„Operation.Name“>
                <wsdl:input message=„Message.Name“ />
                <wsdl:output message=„Message.Name“ />
                <wsdl:fault message=„Message.Name“ />
                …
        </wsdl:operation>
        …
</wsdl:portType>
```

Example:


«Interface»
Interface1
operation1 (inMessage : Message1) : Message2
operation2 ( ) : Message3

```
<wsdl:portType name =„Interface1“>
        <wsdl:operation name=„operation1“>
                <wsdl:input message=„Message1“ />
                <wsdl:output message=„Message2“ />
        </wsdl:operation>
        <wsdl:operation name=„operation2“>
                <wsdl:output message=„Message3“ />
        </wsdl:operation>
</wsdl:portType>
```

The dependencies (from Operation and Document) to the semantic elements are not needed to generate WSDL-code, they will only get used to generate WSDL-S code.

6.1.3 Generation of OWL-S code

To generate OWL-S code from our meta-model we need the ServiceProvider-package and the ProcessFlow-package. The ServiceProvider contains all processes which are the semantic web services as specified in OWL-S. For each process one can declare whether it should be abstract (isAbstract) or not. If the process should be abstract then a SimpleProcess is generated, otherwise an AtomicProcess or CompositeProcess is generated (depends on the process flow model as described below). The properties of the OWL-S Service Profile like serviceName, textDescription and contactInformation can be specified in the ServiceDescriptor (name, description and contact_info) and the attributes serviceParameter, serviceProduct (e.g. a mapping to UNSPSC) or serviceClassification (e.g. a mapping to NAICS) specify the quality guarantees that are provided by a service. The ServiceCategory describes categories of services based on the bases of some classification that may be outside OWL-S and possibly outside OWL. CategoryName is the name of the actual category; taxonomy stores a reference to the taxonomy scheme; value points to a value in the specific taxonomy and code stores to each type of service the code which is associated to the taxonomy.

To describe the functionality of the web service we take a closer look into the ProcessFlow-package. Each Action has Inputs, Outputs, Preconditions and Effects which can be found in the OWL-S member submission with the same names. If the action is an AtomicProcess, then we generate an AtomicProcess, if it is a CallCompositeProcess or consists in a CompositeProcess more than one AtomicProcess then we create a CompositeProcess in OWL-S. Supporting the process flow in a CompositeProcess we use ForkNode and JoinNode to generate Splits and Split/Joins in OWL-S and a combination of DecisionNode and MergeNode to generate a Choice or an If-Then-Else in OWL-S. If there is a simple connection between two AtomicProcesses then we generate a Sequence. Each AtomicProcess points to an operation in the Interfaces-package which allows us to generate the ServiceGrounding (to the WSDL-file). The following table shows a short summary of the elements of our meta-model and their corresponding part in the OWL-S standard. For a detailed transformation, have a look at chapter 7 where the transformation rules have been specified in the openArchitectureWare-language XPand.

Meta-model	**OWL-S**
Process	Simple/Atomic/CompositeProcess
isAbstract = true	SimpleProcess
serviceClassification	serviceClassification
serviceProduct	serviceProduct
serviceParameter	serviceParameter, sParameter
name	serviceName
description	textDescription
contact_info	contactInformation
ServiceCategory	ServiceCategory
categoryName	categoryName
taxonomy	taxonomy
value	value
code	code
Action	Profile
Input	Input
Output	Output
Precondition	Precondition
Effect	Result/Effect
ForkNode, JoinNode	Split / Split-Join
DecisionNode, MergeNode	Choice, If-Then-Else
AtomicProcess	AtomicProcess / Grounding
CallCompositeProcess	CompositeProcess
Connection	Sequence, Any-Order

Table 1: Correspondences between our meta-model and OWL-S

6.2 Transformation to WSMO

The following sections describe how WSMO code can be generated from our meta-model. First the ontology needs to be generated using WSML and our Ontology-package, after that we can generate the WSMO web service. Currently we don't support the modeling of mediators and goals. Both might result in new packages of our meta-model where the former might be more difficult to include and the latter is similar to the outputs and effects already modeled in our ProcessFlow-package.

6.2.1 Generation of the ontology

Each RDFSClass in our Ontology is a concept in the ontology in WSML. Both (our meta-model and WSMO) order classes resp. concepts bottom-up using RDFSsubClassOf in our meta-model (resp. hasSuperConcept in WSML). Each RDFProperty in our ontology is an Attribute in the WSML ontology, whereby the domain of the attribute is already specified. The range of the attribute can be found as RDFSrange-parameter of the RDFProperty. It would also be possible to create a Relation for each RDFProperty which has several parameters then. Each parameter has its own domain and the relations can be ordered using hasSuperRelation (RDFSsubPropertyOf in our meta-model). For a short overview about the corresponding elements, have a look at Table 2. (Note: The non-functional properties for the elements in the ontology have been neglected in the current version of our meta-model)

Meta-model	WSMO
Ontology	ontology
RDFSClass	concept
RDFSsubClassOf	hasSuperConcept
RDFProperty	Attribute
RDFSsubPropertyOf	hasSuperRelation

Table 2: Correspondences between our ontology and the WSMO-ontology

6.2.2 Generation of the WSMO web service

All web services in WSMO are described using their capabilities and interfaces. The capabilities contain preconditions, assumptions, postconditions and effects. They might include shared variables, mediators and ontologies. The interface describes the choreography and orchestration of services building on abstract state machines. Each choreography has states and guarded transitions. These states correspond to our nodes in the ProcessFlow-package and the guarded transitions can be interpreted as our connections. The orchestration is also state-based and consists of the same elements as the choreography, whereby the guarded transitions might have the form *if condition then mediator_uri*. A more detailed description of choreography and orchestration is still missing, why we currently concentrate on the translation of single-step processes. The capability in WSMO describes the functional elements of each service. The inputs of our meta-model can be translated to shared variables which are then used in the preconditions of WSMO, the preconditions of our meta-model are the assumptions in the WSMO capabilities. The outputs are (similar to the inputs) shared variables and used in the postconditions and the effects stay the effects.

The generation of WSMO-code is the only case where the class NonFunctionals of our meta-model is needed. WSMO enables the user to specify much more non-functional descriptions about a web service (or other elements) than the other submissions. Table 3 shows non-functional properties (except for the attributes specified in the class NonFunctional because the names there are identical to the names defined in WSMO) and how they can be interpreted in the WSMO member submission.

Meta-model	**WSMO**
node	state
connection	guarded transition
input	shared variable and precondition
precondition	assumption
output	shared variable and postcondition
effect	effect
name	title
author	creator
contact_info	owner
contributor	contributor
description	description
url	source
identifier	identifier
version	version
releasedate	date
language	language
trust	trust
subject	subject
reliability	reliability
cost	financial

Table 3: Correspondences between our meta-model and WSMO

6.3 Transformation to WSDL-S

To generate WSDL-S code we only need the Interfaces, ServiceProvider and the ProcessFlow-package from our meta-model (no Ontology-package, because WSDL-S stays independent from any ontology-language and therefore the ontology might have been generated using one of the languages introduced above: OWL, WSML or even UML).

WSDL-S uses the extensibility elements of WSDL and introduces two extension attributes (modelReference and schemaMapping), two new elements (precondition and effect) and an extension attribute of the interface element (category). Each operation can now be described with a semantic element (the dependency in our meta-model from Operation to SemanticElement) and all types can be annotated using the same way (Dependency between Document and SemanticElement). Each interface can be described using a category which is bound to the process in our meta-model. Each operation in WSDL is not only described using input and output values, but additionally with preconditions and effects which themselves might have dependencies on semantic elements again. These values from our ProcessFlow-package are used to generate the extensibility elements in WSDL-S.

Meta-model	WSDL-S
Interface	interface
Operation	operation
Dependency to SemanticElement	modelReference
Message	message
ServiceCategory	category
categoryName	categoryname
code	taxonomyCode
taxonomy	taxonomyURI
value	taxonomyValue

Table 4: Correspondences between our meta-model and WSDL-S

6.4 Transformation to SWSF

Each service in SWSF contains a ServiceDescriptor, a Process model and a Grounding. The ServiceDescriptor is the same as in our meta-model and includes all properties to describe the service in more detail. Each AtomicProcess consists of Inputs, Outputs, Preconditions and Effects and might consist of more than one process. There are special categories of processes which might be distinguished: Domain-Specific Atomic process (every Process that is modeled using our meta-model), Produce-Message, Read-Message and Destroy-Message (every AtomicProcess in our meta-model either creates or reads a message) and Channel-Manipulation Atomic Processes (which might modify the properties of a channel). One process in our meta-model corresponds with other processes using one or more channels and these channels deliver the messages to the partner.

Meta-model	SWSF
ServiceDescriptor	ServiceDescriptor
name	name
author	author
contact_info	contact
contributor	contributor
description	description
url	url
identifier	identifier
version	version
releasedate	releaseDate

language	language
trust	trust
subject	subject
reliability	reliability
cost	cost
Channel	channel
AtomicProcess	Domain-specific atomic process combined with a Produce_Message, Read_Message and eventually Destroy_Message
Input	input
Output	output
Precondition	precondition
Effect	effect
ForkNode, JoinNode	Split
DecisionNode, MergeNode	Choice, If-Then-Else
Connection	Sequence, Unordered

Table 5: Correspondences between our meta-model and SWSF

7 XPand transformation rules to OWL-S

To use our profile to work with the standard OWL-S, we need to create transformation rules to the ontology in OWL, the web service in WSDL and the semantic web service description in OWL-S. These transformation rules have been specified using the openArchitectureWare-language XPand. For more details about XPand the interested user might refer to [13].

7.1 XPand rules for OWL

```
«REM»
**********************************************************************
             Transformation rules from our meta-model to OWL
            using the language XPand from openArchitectureWare
                      (www.openarchitectureware.org)

                               Created by
                          Florian Lautenbacher
                          University of Augsburg
                     Programming distributed Systems
                             D-86135 Augsburg
                       www.informatik.uni-augsburg.de/ds
**********************************************************************
These transformations use the package Ontology from our meta-model.
«ENDREM»

«DEFINE Root FOR sytemModel::SWS»
```

```
  «LET "http://www.ds-lab.org/example" AS tns»
  «FILE Ontology.name + „.owl"»
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="«tns»#">
<owl:Ontology rdf:about=""/>
    «EXPAND transformRDFSClass FOREACH RDFSClas»
    «EXPAND transformOWLClass FOREACH OWLClass»
    «EXPAND transformRDFProperty FOREACH RDFProperty»
    «EXPAND transformOWLDatatypeProperty FOREACH OWLDatatypeProperty»
    «EXPAND transformOWLObjectProperty FOREACH OWLObjectProperty»
</rdf:RDF>
  «ENDFILE»
  «ENDLET»
«ENDDEFINE»

«REM»
*********************************************************************
             R D F S C l a s s
«ENDREM»
«DEFINE transformRDFSClass FOR RDFSClass»
<rdfs:Class rdf:ID="«RDFSClass.name»">
«IF RDFSClass.generalization != null»
    <rdfs:subClassOf>
        <rdfs:Class
            rdf:about="#«RDFSClass.generalization.general.name»"/>
    </rdfs:subClassOf>
</rdfs:Class>
«ENDDEFINE»

«REM»
*********************************************************************
             O W L C l a s s
«ENDREM»
«DEFINE transformOWLClass FOR OWLClass»
<owl:Class rdf:ID="«OWLClass.name»">
«IF OWLClass.generalization != null»
    <rdfs:subClassOf>
        <owl:Class
            rdf:about="#«OWLClass.generalization.general.name»"/>
    </rdfs:subClassOf>
</owl:Class>
«ENDIF»
«ENDDEFINE»
```

```
«REM»
*********************************************************************
             R D F P r o p e r t y
«ENDREM»
«DEFINE transformRDFProperty FOR RDFProperty»
<rdf:Property rdf:ID=”«RDFProperty.name»”>
«IF RDFProperty.generalization != null»
    <rdfs:subPropertyOf
     rdf:resource=”«RDFProperty.generalization.general.name»”/>
«ENDIF»
    <rdfs:domain rdf:resource=”«RDFProperty.RDFSdomain.name»”/>
    <rdfs:range rdf:resource=”«RDFProperty.RDFSrange.name»”/>
</owl:DatatypeProperty>
«ENDDEFINE»
«REM»
*********************************************************************
             O W L D a t a t y p e P r o p e r t y
«ENDREM»
«DEFINE transformOWLDatatypeProperty FOR OWLDatatypeProperty»
<owl:DatatypeProperty rdf:ID=”«OWLDatatypeProperty.name»”>
«IF OWLDatatypeProperty.generalization != null»
    <rdfs:subPropertyOf
     rdf:resource=”«OWLDatatypeProperty.generalization.general.name»”/>
«ENDIF»
    <rdfs:domain rdf:resource=”«OWLDatatypeProperty.RDFSdomain.name»”/>
    <rdfs:range rdf:resource=”«OWLDatatypeProperty.RDFSrange.name»”/>
</owl:DatatypeProperty>
«ENDDEFINE»
«REM»
*********************************************************************
             O W L O b j e c t P r o p e r t y
«ENDREM»
«DEFINE transformOWLObjectProperty FOR OWLObjectProperty»
<owl:ObjectProperty rdf:ID=”«OWLObjectProperty.name»”>
«IF OWLObjectProperty.generalization != null»
    <rdfs:subPropertyOf
     rdf:resource=”«OWLObjectProperty.generalization.general.name»”/>
«ENDIF»
    <rdfs:domain rdf:resource=”«OWLObjectProperty.RDFSdomain.name»”/>
    <rdfs:range rdf:resource=”«OWLObjectProperty.RDFSrange.name»”/>
</owl:ObjectProperty>
«ENDDEFINE»
```

7.2 XPand rules for WSDL

```
«REM»
**********************************************************************
           Transformation rules from our meta-model to WSDL 1.1
            using the language XPand from openArchitectureWare
                      (www.openarchitectureware.org)

                               Created by
                          Florian Lautenbacher
                         University of Augsburg
                     Programming distributed Systems
                            D-86135 Augsburg
                      www.informatik.uni-augsburg.de/ds
**********************************************************************
These transformations use the package Interface from our meta-model (not
to be confused with the class Interface in the same package!). First,
all documents are generated, after that all messages and operations and
based on that, all interfaces are generated.
«ENDREM»

«DEFINE Root FOR sytemModel::SWS»
  «LET "http://www.ds-lab.org/example" AS tns»
  «FILE Interface.name + „.wsdl"»
<wsdl:definitions
     name="«Interface.name»"
     targetNamespace=«tns»>
  <wsdl:types>
    <xsd:schema>
      «EXPAND transformDocument FOREACH Interface::Document»
    </xsd:schema>
  </wsdl:types>
      «EXPAND transformMessage FOREACH Interface::Message»
      «EXPAND transformPortType FOREACH Interface::Interface»
     «REM»
         Transformation  rules  for  the  binding  and  description  of  the
         services might be added here.
    «ENDREM»
</wsdl:definitions>
  «ENDFILE»
  «ENDLET»
«ENDDEFINE»


«REM»
*********************************************************************
             D o c u m e n t
«ENDREM»
«DEFINE transformDocument FOR Document»
     <complexType name="«Document.name»">
        <all>
         «EXPAND transformAttribute FOREACH Document.attribute»
```

```
       </all>
    </complexType>
    <element name="«Document.name»“ type="tns:«Document.name»“/>
«ENDDEFINE»

«DEFINE transformAttribute FOR Attribute»
    <element name="«Attribute.name»“
        «IF Attribute.type == “String”»
         type = “String”/>
        «ELSEIF Attribute.type == “Integer”»
         type = “int”/>
        «ELSEIF Attribute.type == “Boolean”»
         type = “Boolean”/>
        «ELSEIF Attribute.type == “Real”»
         type = “Real”/>
        «REM»
            The primitive datatypes don’t need a namespace, only the
            complex ones.
        «ENDREM»
        «ELSE»
         type="«tns»:«Attribute.type»”/>
        «ENDIF»
«ENDDEFINE»

«REM»
*********************************************************************
             M e s s a g e
«ENDREM»
«DEFINE transformMessage FOR Message»
   <wsdl:message name="«Message.name»“>
    «EXPAND transformMessageDocumentAttribute FOREACH
        Message.document»
   </wsdl:message>
«ENDDEFINE»

«DEFINE transformMessageDocumentAttribute FOR Document»
    <wsdl:part name="«Attribute.name»“
     element="«tns»:«Document.name»“/>
«ENDDEFINE»

«REM»
*********************************************************************
             P o r t T y p e
«ENDREM»
«DEFINE transformPortType FOR Interface»
  <wsdl:portType name="«Interface.name»“>
       «EXPAND transformOperation FOREACH Interface.operation»
  </wsdl:portType>
«ENDDEFINE»

«DEFINE transformOperation FOR Operation»
  <wsdl:operation name="«Operation.name»">
    «IF Operation.in != null»
        <wsdl:input message="«Operation.in.name»"/>
    «ENDIF»
    «IF Operation.out != null»
        <wsdl:output message="«Operation.out.name»"/>
    «ENDIF»
    «FOREACH Operation.fault AS fault»
        <wsdl:fault message="«Operation.fault.name»"/>
    «ENDFOREACH»
  </wsdl:operation>
«ENDDEFINE»
```

7.3 XPand rules for OWL-S

```
«REM»
**********************************************************************
          Transformation rules from our meta-model to OWL-S 1.2
           using the language XPand from openArchitectureWare
                     (www.openarchitectureware.org)

                              Created by
                         Florian Lautenbacher
                        University of Augsburg
                    Programming distributed Systems
                           D-86135 Augsburg
                   www.informatik.uni-augsburg.de/ds
**********************************************************************
These transformations use all packages from our meta-model to generate
the service profile, service model and service grounding.
«ENDREM»
«DEFINE Root FOR sytemModel::SWS»
  «LET "http://www.ds-lab.org/example" AS tns»
  «FILE Process.name + „Service.owl"»
<rdf:RDF xml:base="«tns»/«Process.name»Service.owl">
    «EXPAND transformOntologyImport FOR Process»
    <service:Service rdf:ID="«Process.name»">
        <service:presents
            rdf:resource="./«Process.name»Profile.owl"/>
        <service:describedBy
            rdf:resource="./«Process.name»Process.owl"/>
        <service:supports
            rdf:resource="./«Process.name»Grounding.owl"/>
    </service:Service>
</rdf:RDF>
  «ENDFILE»
  «ENDLET»
    «EXPAND transformProcess FOREACH Process»
    «EXPAND transformProfile FOREACH Process»
    «EXPAND transformGrounding FOREACH Process»
«ENDDEFINE»

«REM»
*********************************************************************
             P r o c e s s
«ENDREM»
«DEFINE transformProcess FOR Process»
  «FILE Process.name + „Process.owl“»
<rdf:RDF xml:base=”«tns»/«Process.name»Process.owl”>
    «EXPAND transformOntologyImport FOR Process»
    «REM»
        If there is only one action modeled, then it is an AtomicProc-
        ess (or Simple-AtomicProcess), if there are more nodes, then it
        is a CompositeProcess (or a Simple-CompositeProcess)
    «ENDREM»
    «IF Process.hasBehavior.node.size > 1»
        «EXPAND transformAtomicProcess FOR Process»
    «ELSE»
        «EXPAND transformCompositeProcess FOR Process»
    «ENDIF»
</rdf:RDF>
  «ENDFILE»
«ENDDEFINE»

«REM»
*********************************************************************
             O n t o l o g y I m p o r t
«ENDREM»
«DEFINE transformOntologyImport FOR Process»
    <owl:Ontology rdf:about=””>
        <owl:imports>
            <owl:Ontology
         rdf:about=http://www.daml.org/services/owl-s/1.2/Service.owl/>
        </owl:imports>
        <owl:imports>
            <owl:Ontology
         rdf:about=http://www.daml.org/services/owl-s/1.2/Process.owl/>
        </owl:imports>
        <owl:imports>
            <owl:Ontology
                rdf:about=http://www.daml.org/services/owl-
                s/1.2/ProfileHierarchy.owl/>
        </owl:imports>
        «REM»
```

```
             The following imports the OWL-ontology that has been
             generated before.
         «ENDREM»
         <owl:imports>
             <owl:Ontology
                 rdf:about=«tns»/«Ontology.Ontology.name».owl”/>
         </owl:imports>
     </owl:Ontology>
«ENDDEFINE»

«REM»
*********************************************************************
             A t o m i c P r o c e s s
«ENDREM»
«DEFINE transformAtomicProcess FOR Process»
«IF Process.isAbstract == ”true”»
    <process:SimpleProcess rdf:ID=”«Process.name»”>
«ELSE»
    <process:AtomicProcess rdf:ID=”«Process.name»”>
«ENDIF»
    «EXPAND transformInput FOREACH
         Process.hasBehavior.node.first.input»
    «EXPAND transformOutput FOREACH
         Process.hasBehavior.node.first.output»
    «EXPAND transformPrecondition FOREACH
        Process.hasBehavior.node.first.localPrecondition»
    «EXPAND transformEffect FOREACH
        Process.hasBehavior.node.first.effect»
«ENDIF»
«ENDDEFINE»

«REM»
*********************************************************************
             I n p u t
«ENDREM»
«DEFINE transformInput FOR Input»
<process:hasInput>
    <process:Input rdf:ID=”«Input.Name»”>
        <process:parameterType
            rdf:datatype=http://www.w3.org/2001/XMLSchema#anyURI>
            «Input.depends.type»”
        </process:parameterType>
    </process:Input>
</process:hasInput>
«ENDDEFINE»

«REM»
```

```
*********************************************************************
             O u t p u t
«ENDREM»
«DEFINE transformOutput FOR Output»
<process:hasOutput>
    <process:Output rdf:ID=”«Output.Name»”>
        <process:parameterType
            rdf:datatype=http://www.w3.org/2001/XMLSchema#anyURI>
            «Input.depends.type»”
        </process:parameterType>
    </process:Output>
</process:hasOutput>
«ENDDEFINE»

«REM»
*********************************************************************
             P r e c o n d i t i o n
«ENDREM»
«DEFINE transformPrecondition FOR Precondition»
<process:hasPrecondition>
«REM»
    Preconditions and Effect currently need to be specified in a form
    that they can be processed by OWL-S (e.g. directly in SWRL). There-
    fore, at the moment, we only copy the content of the Expressions
    here.
«ENDREM»
«Precondition.expression.content»
</process:hasPrecondition>
«ENDDEFINE»

«REM»
*********************************************************************
             E f f e c t
«ENDREM»
«DEFINE transformEffect FOR Effect»
<process:hasEffect>
«REM»
    Preconditions and Effect currently need to be specified in a form
    that they can be processed by OWL-S (e.g. directly in SWRL). There-
    fore, at the moment, we only copy the content of the Expressions
    here.
«ENDREM»
«Effect.expression.content»
</process:hasEffect>
«ENDDEFINE»

«REM»
*********************************************************************
             C o m p o s i t e P r o c e s s
```

```
For the transformation of composite processes, one needs several Java
methods to get from one node to the next one. More detailed, one needs
the methods getFirstElement(), getNextElement(), etc.
«ENDREM»
«DEFINE transformCompositeProcess FOR Process»
«IF Process.isAbstract == ”true”»
    <process:SimpleProcess rdf:ID=”«Process.name»”>
«ELSE»
    <process:CompositeProcess rdf:ID=”«Process.name»”>
«ENDIF»
«EXPAND transformSequence FOR Process.getFirstElement()»
«ENDDEFINE»

«REM»
*******************************************************************
                            S e q u e n c e
Sequence: Ablauf von Prozessen, die bereits in einer Sequenz enthalten
sind. transformSequence wird nur aufgerufen, wenn der momentane Knoten
eine Aktion oder ein InitialNode ist.
«ENDREM»
«DEFINE transformSequence FOR Node»
  «LET Process.getNextElement(Node) AS nextelem»
  «IF Node.Type == Action.Type»
       «EXPAND transformAction FOR Node»
  «ENDIF»
  «IF nextelem.Type == Action.Type»
       «EXPAND transformSequence FOR nextelem»
  «ELSEIF nextelem.Type == MergeNode.Type»
       «EMPTY»
  «ELSEIF nextelem.Type == JoinNode.Type»
       «EMPTY»
  «ELSEIF nextelem.Type == ForkNode.Type»
       «EXPAND transformFlow FOR nextelem»
       «EXPAND transformSequence FOR nextelem»
  «ELSEIF nextElem.Type == DecisionNode.Type»
       «EXPAND transformFlow FOR nextelem»
       «EXPAND transformSequence FOR nextelem»
  «ELSE»
       «EXPAND transformFlow FOR nextelem»
  «ENDIF»
  «ENDLET»
«ENDDEFINE»

«REM»
*******************************************************************
                            F l o w
«ENDREM»
«DEFINE transformFlow FOR Node»
  «IF Node.Type == Action.Type»
       <sequence>
              «EXPAND transformSequence FOR Node»
       </sequence>
  «ELSEIF Node.Type == FinalNode.Type»
       «EXPAND transformTerminate FOR Node»
  «ELSEIF Node.Type == ForkNode.Type»
       <flow>
              «EXPAND transformFlow FOREACH
                     Process.getNextElements(Node)»
```

```
        </flow>
   «ELSEIF Node.Type == DecisionNode.Type»
        <switch>
               «EXPAND transformCase FOREACH DecisionNode.outgoing»
        </switch>
   «ELSEIF Node.Type == MergeNode.Type»
        «EXPAND transformFlow FOREACH
                      Process.getNextElement(Node)»
   «ELSEIF Node.Type == JoinNode.Type»
        «EXPAND transformFlow FOREACH
                      Process.getNextElement(Node)»
   «ENDIF»
«ENDDEFINE»

«REM»
********************************************************************
                            C a s e
«ENDREM»
«DEFINE transformCase FOR Connection»
   <case condition="«ProcessFlow.getGuard(Connection)»">
        «EXPAND transformFlow FOR
               ProcessFlow.getNextElement(Connection)»
   </case>
«ENDDEFINE»

«REM»
********************************************************************
                        T e r m i n a t e
«ENDREM»
«DEFINE transformTerminate FOR Node»
   <terminate/>
«ENDDEFINE»

«REM»
********************************************************************
                            A c t i o n
«ENDREM»
«DEFINE transformAction FOR Action»
     «IF Action.Type == AtomicProcess»
          «EXPAND transformAtomicProcess FOR Action.target»
     «ELSEIF Action.Type == CallCompositeProcess»
          «EXPAND transformCompositeProcess FOR Action.behaviour»
     «ENDIF»
«ENDDEFINE»
«REM»
********************************************************************
                          P r o f i l e
«ENDREM»
«DEFINE transformProfile FOR Process»
  «FILE Process.name + „Profile.owl“»
<rdf:RDF xml:base="«tns»/«Process.name»Profile.owl">
     «EXPAND transformOntologyImport FOR Process»
     <service:presentedBy rdf:resource="./«Process.name»Service.owl"/>
     <profile:has_process rdf:resource="./«Process.name»Process.owl"/>
```

```
    <profile:serviceName>
        «Process.serviceDescriptor. name»
    </profile:serviceName>
    <profile:textDescription>
        «Process.serviceDescriptor.description»
    </profile:textDescription>
    <profile:contactInformation>
        «Process.serviceDescriptor.contact_info»
    </profile:contactInformation>
    «EXPAND transformInputProfile FOREACH
        Process.hasBehavior.node.first.input»
    «EXPAND transformOutputProfile FOREACH
        Process.hasBehavior.node.last.output»
    «EXPAND transformPreconditionProfile FOREACH
        Process.hasBehavior.node.first.precondition»
    «EXPAND transformEffectProfile FOREACH
        Process.hasBehavior.node.last.effect»
</rdf:RDF>
  «ENDFILE»
«ENDDEFINE»

«REM»
*********************************************************************
             I n p u t P r o f i l e
«ENDREM»
«DEFINE transformInputProfile FOR Input»
    <profile:hasInput rdf:resource=
    "./«Process.name»Process.owl#«Input.name»"/>
«ENDDEFINE»

«REM»
*********************************************************************
             O u t p u t P r o f i l e
«ENDREM»
«DEFINE transformOutputProfile FOR Output»
    <profile:hasOutput rdf:resource=
    "./«Process.name»Process.owl#«Output.name»"/>
«ENDDEFINE»

«REM»
*********************************************************************
             P r e c o n d i t i o n P r o f i l e
«ENDREM»
«DEFINE transformPreconditionProfile FOR Precondition»
    <profile:hasPrecondition rdf:resource=
    "./«Process.name»Process.owl#«Precondition.name»"/>
«ENDDEFINE»
```

```
«REM»
*********************************************************************
             E f f e c t P r o f i l e
«ENDREM»
«DEFINE transformEffectProfile FOR Effect»
    <profile:hasResult rdf:resource=
    ”./«Process.name»Process.owl#«Effect.name»”/>
«ENDDEFINE»

«REM»
*********************************************************************
             G r o u n d i n g
Each interface, operation, message and document needs to be grounded to
the generated WSDL-file.
«ENDREM»
«DEFINE transformGrounding FOR Process»
  «FILE Process.name + „Grounding.wsdl“»
    <?xml version=”1.0”?>
    <definitions name=”«Process.name»_WSDL”
    targetNamespace=”«tns»/«Process.name»Grounding.wsdl”
    xmlns:owl-s-wsdl=”http://www.daml.org/services/owl-s/wsdl”
    xmlns=”http://schemas.xmlsoap.org/wsdl/”>
    «EXPAND transformMessageGrounding FOREACH Operation»
    «EXPAND transformInterfaceGrounding FOREACH Interface»
    </definitions>
  «ENDFILE»
«ENDDEFINE»
«REM»
*********************************************************************
             M e s s a g e G r o u n d i n g
«ENDREM»
«DEFINE transformMessageGrounding FOR Operation»
    «IF Operation.input != null»
    <message name=”«Operation.name»_Input”>
        <part name=”«Operation.input.name»”
                owl-s-wsdl:owl-s-parameter=”«Process.name»:#Input”/»
    </message>
    «ENDIF»
    «IF Operation.output != null»
    <message name=”«Operation.name»_Output”>
        <part name=”«Operation.output.name»”
                owl-s-wsdl:owl-s-parameter=”«Process.name»:#Output”/»
    </message>
    «ENDIF»
«ENDDEFINE»
```

```
«REM»
**********************************************************************
             I n t e r f a c e G r o u n d i n g
«ENDREM»
«DEFINE transformInterfaceGrounding FOR Interface»
    <portType name=”«Interface.name»_PortType”>
        «EXPAND transformOperationGrounding FOREACH Operation»
    </portType>
«ENDDEFINE»

«REM»
**********************************************************************
             O p e r a t i o n G r o u n d i n g
«ENDREM»
«DEFINE transformOperationGrounding FOR Operation»
    <operation name=”«Operation.name»_operation
        owl-s-wsdl:owl-s-process=”«Process.name»:#«Operation.name»>
        «EXPAND transformInputGrounding FOREACH Input»
        «EXPAND transformOutputGrounding FOREACH Output»
    </operation>
«ENDDEFINE»

«REM»
**********************************************************************
             I n p u t G r o u n d i n g
«ENDREM»
«DEFINE transformInputGrounding FOR Input»
    <input message=”tns:«Operation.input.name»_Input”/>
«ENDDEFINE»

«REM»
**********************************************************************
             O u t p u t G r o u n d i n g
«ENDREM»
«DEFINE transformOutputGrounding FOR Output»
    <output message=”tns:«Operation.output.name»_Output”/>
«ENDDEFINE»
```

8 Related Work

There are several efforts to create a UML profile for Semantic Web Services. However, to our knowledge none of the existing approaches tries to consider every existing W3C submission of semantic web services (meaning OWL-S, WSMO, WSDL-S and SWSF).

In [18, 19] Roy Grønmo et al. define transformations between UML and OWL-S and a web service composition based on these information. The developed profile uses the UML Ontology Profile (defined by Duric for UML 1.5 class diagrams) to model the concepts of the ontology. They use a UML activity to describe a web service and attaching inputs and outputs which makes it difficult to use control nodes for the composition of several web services later. Their profile supports the generation of OWL-S and WSMO code (hence, there are no transformation rules for the generation of WSMO), but doesn't consider SWSF and WSDL-S.

In [20] (which received a best student paper award) a model-driven approach for specifying semantic web services has been developed. However, the UML-profile only considered AtomicProcesses in OWL-S, not including the collaboration of several processes. It is only applicable to OWL-S and misses transformation rules for WSMO, SWSF and WSDL-S.

[21] describes how OWL-S services can be modeled, but in a proprietary format not using the UML-profiling mechanism. [22] develops an MDD annotation methodology for semantic enhanced SOAs, but does not develop a UML profile for semantic web services in greater detail. [23] describes a case study with a methodological framework for the development of semantic web information systems (MIDAS-S) building on WSMO. In [24] (and other talks) E. Kendall promotes the integration of OWL-S and SWSF within the ODM. We completely agree and support this initiative, if the meta-model considers the other approaches of semantic web services named WSMO and WSDL-S, too.

9 Conclusions and Future Investigations

Using our meta-model and UML-profile one can simply model a semantic web service and then generate code in one of the currently proposed SWS-languages. Our profile provides independency from each single SWS standard and can easily be adapted in the future. We integrated our profile into a UML CASE-tool and modeled a well-known example. Additionally, we showed how code can be generated using an informal description as well as the openArchitectureWare format XPand.

Our meta-model fulfills the following requirements:

- it is easy-to-understand and easy-to-use
- it is compatible with all of the current W3C SWS-submissions and builds on the OMG specification draft
- is enables a code generation through a well-defined meta-model
- it includes the modeling of ontologies

However, rules are still missing. To make it easier to model preconditions and effects we will include rules (based on SWRL and WRL) in upcoming versions of our profile. We are also interested on semantic business process models and how to combine these business processes with the developed meta-model for semantic web services.

Appendix A: OWL-S meta-model

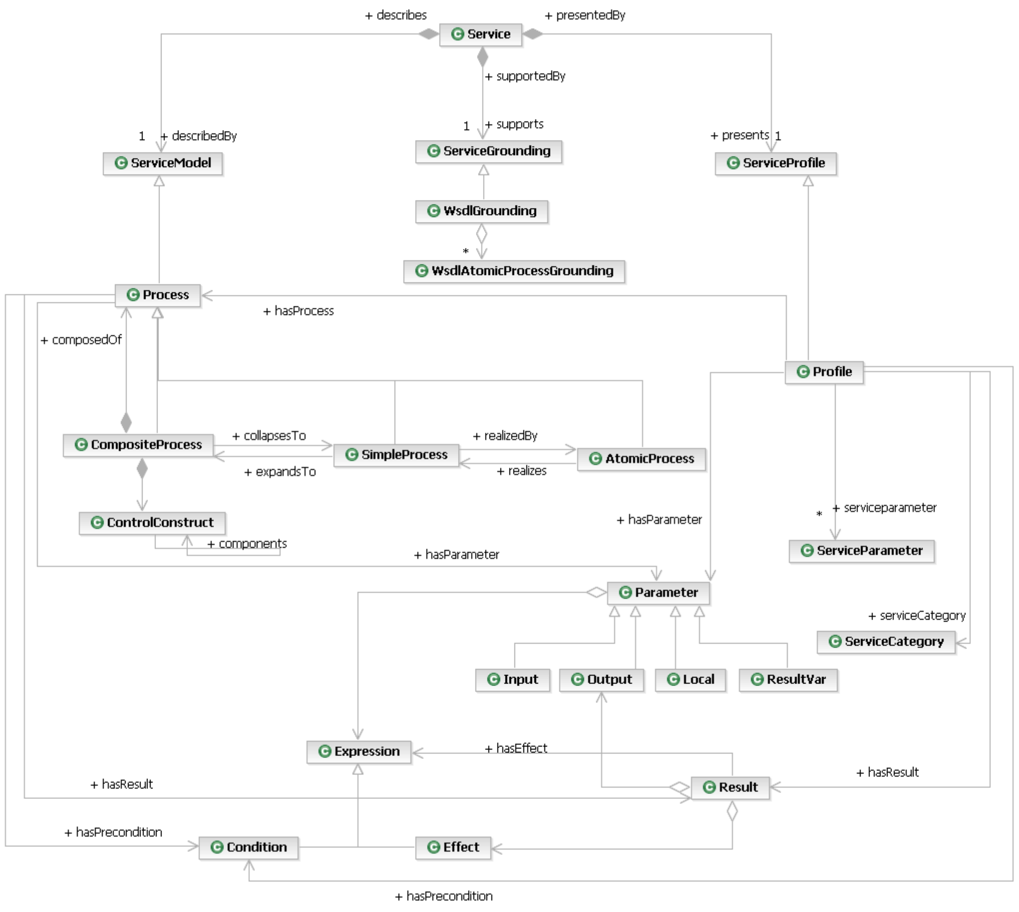


Figure 37: OWL-S meta-model

References

1. Object Management Group (OMG): “Unified Modeling Language (UML) Specification: Superstructure, Version 2.0, Final Adopted Specification”, July 2005, available online at http://www.omg.org/docs/formal/05-07-04.pdf
2. Object Management Group (OMG): “Unified Modeling Language (UML) Specification: Infrastructure, Version 2.0, Final Adopted Specification”, July 2005, available online at http://www.omg.org/docs/formal/05-07-05.pdf
3. Object Management Group (OMG): “Meta Object Facility (MOF) Core Specification, Version 2.0”, January 2006, available online at http://www.omg.org/docs/formal/06-01-01.pdf
4. Object Management Group (OMG): “MDA Guide Version 1.0.1”, June 2003, available online at http://www.omg.org/docs/omg/03-06-01.pdf
5. Object Management Group (OMG): “Ontology Definition Metamodel (ODM), Fifth Revised Submission to OMG/RFP ad/2003-03-40”, January 2006, available online at http://www.omg.org/docs/ad/06-01-01.pdf
6. Martin, D. et al: “OWL-S: Semantic Markup for Web Services”, November 2004, W3C Member Submission, available online at http://www.w3.org/Submission/OWL-S/
7. Lausen, H., Polleres, A. and Roman, D. (Eds.): “Web Service Modeling Ontology (WSMO)”, June 2005, W3C Member Submission, available online at http://www.w3.org/Submission/WSMO/
8. Akkiraju, R. et al.: “Web Service Semantics – WSDL-S”, November 2005, W3C Member Submission, available online at http://www.w3.org/Submission/WSDL-S/
9. Battle, S. et al.: “Semantic Web Services Framework (SWSF) Overview”, September 2005, W3C Member Submission, available online at http://www.w3.org/Submission/SWSF/
10. Hassael-Massieux, D. and Connolly, D.: “Gleaning Resource Descriptions from Dialects of Languages (GRDDL)”, May 2005, W3C Team Submission, available online at http://www.w3.org/TeamSubmission/grddl/
11. Kopecky, J. and Parsia, B.: “Web Service Description Language (WSDL) Version 2.0: RDF Mapping”, May 2006, W3C Working Draft, available online at http://www.w3.org/TR/wsdl20-rdf/
12. MID Enterprise Software Solutions: InnovatorAOX 2006, information online at http://www.mid.de/
13. Efftinge, S. and Kadura, C.: “OpenArchitectureWare 4.1 Xpand Language Reference”, available online at http://www.eclipse.org/gmt/oaw/doc/4.1/r20_xPandReference.pdf
14. Alves, A. et al.:” Web Services Business Process Execution Language Version 2.0”, Committee Draft, May 2006, available online at http://www.oasis-open.org/committees/download.php/18714/wsbpel-specification-draft-May17.htm
15. Workflow Management Coalition Specification: “XML Process Definition Language”, WFMC-TC-1025, October 2005, available online at http://www.wfmc.org/standards/docs/TC-1025_xpdl_2_2005-10-03.pdf
16. van der Aalst, W.M.P. and Dumas, M. and ter Hofstede A.H.M. and Wohet, P.: “Pattern Based Analysis of BPML (and WSCI)”, FIT Technical Report, 2002, available online at http://xml.coverpages.org/Aalst-BPML.pdf
17. Business Process Management Initivative (BPMI.org): “Business Process Modeling Notation (BPMN)”, Version 1.0, May 2004, available online at http://www.bpmn.org/Documents/BPMN V1-0 May 3 2004.pdf

18. Skogan, D. and Gronmo, R. and Solheim, I.: “Web Service Composition in UML”, presented at the 8th International Enterprise Distributed Object Computing Conference (EDOC), Monterey, September 2004
19. Gronmo, R. and Jaeger, M. and Hoff, H.: “Transformations between UML and OWL-S”, presented at the European Conference on Model Driven Architecture – Foundations and Applications (ECMDA-FA), Nuremberg, November 2005
20. Timm, J. and Gannod, G.: “A Model-Driven Approach for Specifying Semantic Web Services”, presented at the 3rd IEEE International Conference on Web Services (ICWS 2005), July 2005
21. Scicluna, J. and Abela, C. and Montebello, M.: “Visual Modeling of OWL-S Services”, Proceedings of the second Computer Science Annual Workshop (CSAW’04) at the University of Malta, September 2004.
22. Pondrelli L.: “An MDD annotation methodology for Semantic Enhanced Service Oriented Architectures”, Proceedings of the Open Interop Workshop on Enterprise Modelling and Ontologies, Porto, June 2005.
23. Acuna, C. and Marcos, E.: “Modeling semantic web services – a case study”, Proceedings of the 6th International conference on web engineering (ICWE06), Palo Alto, CA, USA, 2006.
24. Kendall, E.: “MDA and Semantic Web Services: Integrating OWL-S & SWSF with the Ontology Definition Metamodel (ODM)”, SOA, MDA and Web Services Workshop, OMG, March 2006.
25. Peer, J.: “Semantic Service Markup with SESMA”, Web Service Semantics Workshop (WSS’05) at the World Wide Web Conference ’05, 2005.
26. Fensel, D. and Bussler, C.: “The Web Service Modeling Framework WSMF”, Electronig Commerce: Research and Applications, 2002
27. de Bruijn, J. and Lausen, H. and Polleres, A. and Fensel, D.: “WSML – a Language Framework for Semantic Web Services”, W3C Workshop on Rule Languages for Interoperability, Washington DC, USA, 2005.
28. Lautenbacher, F. and Bauer, B.: “Semantic Reference and Business Process Modeling Enables an Automatic Synthesis”, Semantics for Business Process Management (SBPM) at European Semantic Web Conference (ESWC06), Budva, Montenegro, June 2006.
29. Pistore, M. and Spalazzi, L. and Traverso, P.: “A Minimalist Approach to Semantic Annotations for Web Processes Compositions”, European Semantic Web Conference (ESWC06), Budva, Montenegro, June 2006.
30. Farell, J. and Lausen, H.: “Semantic Annotations for WSDL (SAWSDL) – W3C Working Draft 28 September 2006”, available online at http://www.w3.org/TR/sawsdl/.