Semantic Model-Driven Architecting of Service-based Software Systems

Claus Pahl
Dublin City University
School of Computing
Dublin 9, Ireland
email: Claus.Pahl@dcu.ie
phone: ++353 +1 700 5620
fax: ++353 +1 700 5442

ABSTRACT:
Model-driven development is a software development framework that emphasises model-based abstraction and automated code generation. Service-based software architectures benefit in particular from semantic, ontology-based modelling. We present ontology-based transformation and reasoning techniques for layered semantic service architecture modelling. Integrated ontological layers support abstract domain modelling, architectural design, and interoperability aspects. Ontologies are beneficial due to their potential to formally define models, to allow reasoning about semantic models, and to automate transformations at all layers. Ontologies are suitable in particular for the Web Services platform due to their ubiquity within the Semantic Web and their application to support semantic Web services.

KEYWORDS:
Software Architecture, Model-Driven Development, Web Services, Semantic Modelling, Ontologies.

1 Introduction

The recognition of the importance of modelling as a means of abstraction and the need to automate software development has over the past years led to model-driven development as a software engineering approach [19]. Model-driven development combines layered modelling techniques based on notations such as the Unified Modelling Language (UML) with automated transformations and code generation. Recently, ontology-based modelling has been investigated as a semantic modelling framework that enhances the semantic richness of the classical UML-based approaches. While formal modelling and reasoning is, to some extent, available in the UML context in form of the Object Constraint Language OCL, ontologies as logic-based formalisms offer full reasoning support. A second benefit of ontologies as sharable knowledge representations is the potential to easily reuse and share models.

Modelling and developing software systems as service-based architectures is gaining increasing momentum [2,5,18]. The activities of modelling and describing these reusable and composable services is central for both providers and clients of services. Providers need to provide an accurate description or model for a service that can be inspected by potential clients. In particular the attention that Web services [32] have received recently emphasises the importance of service-orientation as the architectural paradigm. Service-oriented architecture is becoming an important software engineering paradigm. Our focus is the Web services platform based on techniques such as the service description language WSDL and the service invocation protocol SOAP, but also extensions like the service-based business process execution language WS-BPEL [33,27,14]. This specific area is particularly suitable to demonstrate the benefits of semantic ontology-based modelling due to the component-orientation and distributed nature of service-based software development with its emphasis on provision and discovery of descriptions and on sharing and reusing of models and services. In

addition to the modelling capabilities, ontologies also provide a formal framework that enables reasoning and transformation and, thus, supports the automation of development activities.

While a process has started towards the development of an ontology metamodel that can support semantic, ontology-based modelling for platform-independent model layers [20], we take a more comprehensive approach here with ontologies for computation-independent, platform-independent, and platform-specific layers. Other authors, e.g. [10], have already explored Web ontology languages to enhance modelling capabilities in model-driven development frameworks. We extend these approaches by presenting here a layered, ontology transformation-based semantic modelling approach for software services. Ontologies turn out to support a number of modelling tasks – from domain modelling to architectural configuration and also service and process interoperability. We put an emphasis on processes, which play an important role in modelling domain activities such as business processes, but also in modelling interaction processes in software architecture configurations. Process-orientation and interaction and composition in distributed architectures are central for service-based software systems.

Our work has to be seen in the context of three fundamental paradigms of software and knowledge engineering – model-driven development, service-oriented architecture, and ontology. While a unification of these three paradigms is not our goal, our aim is to adapt model-driven development to service architecture using ontology technology as the integrating tool. We propose a framework for ontology-based model-driven architecture of service-based software systems, i.e. model-driven architecture as the development approach, applied here to service-oriented architecture as the platform and ontology as the semantic modelling technique. Our contribution is a layered ontological transformation framework with different ontologies focussing on the needs of modelling of services and processes at particular abstraction layers.

We start with an overview of ontology-based service development in Section 2. Layered service modelling with ontologies is introduced in Section 3. We address the transformations between ontological layers in Section 4. We discuss our efforts in the context of interoperability and standardisation activities in Section 5. Related work is discussed in Section 6. We end with some conclusions in Section 7.

2 The Development of Service Architectures

The Web services platform defines a Web service as a software system that provides a coherent set of operations at a certain location [32]. The service provider makes an abstract service interface description available that can be used by potential service users to locate and invoke the service using XML-based messages. Services have so far usually been used ’as is’ in single request-response interactions. However, the configuration and coordination of services in service-based architectures and the composition of services to processes is equally important in the second generation of service technology. Existing services can be reused and composed to form business or workflow processes.

Model-driven architecture (MDA) is a model-driven development framework emphasising the importance of modelling for the design of software systems and their architecture, which is promoted by the Object Management Group [19]. The need to address abstract semantic descriptions and to address composition in the context of service-based software development makes MDA (in combination with ontologies) a suitable framework. MDA is based on a three-layered approach:

- The Computation Independent Model (CIM) describes a system from the computation-independent viewpoint, addressing structural aspects of the system. A CIM is often called a domain model.
- The Platform Independent Model (PIM) can be seen as defining a system in terms of a technology-neutral virtual machine or a computational abstraction.
- The Platform Specific Model (PSM) usually consists of a platform model that captures the technical concepts and services that make up the platform and an implementation-specific model geared towards the concrete implementation technique.

Although platform-neutral by definition, the archetypical MDA is based on UML for platform independent modelling and one of the predominant component technologies (EJB, .NET, CORBA) as the platform. In our context, the platform is service-based. The Web services discovery and invocation infrastructure – a directory or marketplace where potential users can search for suitable services and an invocation protocol – with the services and their clients form our platform, i.e. a service-oriented architecture. Different platform types can be distinguished. The generic platform is service-oriented architecture here, the technology-specific platform is the Web services platform, and vendor-specific platform technologies include for instance the Apache Axis or Oracle BPEL service engines.

Description languages are central elements of service-oriented architecture. With the second generation of service technology and efforts such as MDA, the emphasis has shifted from description to the wider and more comprehensive activity of modelling. Behaviour and interaction processes are essential parts of modelling and understanding software architectures [1,27].

Ontology languages – the backbone of the Semantic Web – are knowledge representation and logical inference techniques [6]. They can create a precisely defined shared understanding of annotations of resources such as Web pages or services. Ontologies usually consist of hierarchical definitions of important concepts in a domain and descriptions of the properties of each concept, supported by logics for knowledge representation and reasoning. Ontologies are, however, important beyond sharable and processable annotations of Web resources. Some effort has already been made to exploit Semantic Web and ontology technology for the software engineering domain in general and modelling in particular [24]. OWL-S [7] is a service ontology, i.e. it is a language that provides a specific vocabulary for describing properties and capabilities of Web services, which shows the potential of this technology for service engineering. Formality in the Semantic Web framework facilitates machine understanding and automated reasoning. A variant of the Web Ontology Language OWL, called OWL-DL, is equivalent to description logics [3]. This fruitful connection provides well-defined semantics and reasoning systems. Description logic is particularly interesting for the software engineering context due to a correspondence between description logic and dynamic logic (a modal logic of programs), which has been used to model and reason about software systems [3,30].

3 Modelling with Ontologies

MDA proposes three modelling layers – each with a distinct focus that, as we aim to demonstrate, can be supported ontologically for service-based software development. The computation-independent layer focuses on domain capture. The platform-independent layer focuses on architecture configuration and service process composition. The platform-specific layer focuses on interoperability and discovery support. A case study from the banking domain will accompany our discussion of the three layers.

3.1 Models and Ontologies

The notions of models and ontologies need to be clarified before we can address the individual MDA layers.

- Models in the software development process are constructed to specify, visualise, and document software artefacts and their properties. A common modelling language is the Unified Modelling Language UML. It addresses the conceptual modelling of structural and behavioural properties of software systems [6]. Often, the purpose of model construction is also to enable reasoning within a logical framework, i.e. a stronger semantic modelling considering the semantics in a formal framework.
- An ontology is the formulation of a conceptualisation of a domain – usually hierarchically structured based on subsumption (classification) relationships, but also other semantic relationship types such as composition [6]. The purpose of ontologies is to enable classification and reasoning. Ontology languages allow conceptual modelling based on the introduction of vocabularies and taxonomies, but they also provide a logical framework to specify rules and to reason about expressions.

In particular semantic modelling and ontologies are similar in their purpose. Logic-based ontology languages are suitable to enhance traditional modelling languages in order to enable model-driven service architecture. We propose therefore ontology-based semantic modelling to support model-driven architecting of service-based software systems.

UML 2.0 allows the description of conceptual models where subclass relationships can be semantically specified. The subclass relationship is also semantically well specified at the meta-level. The building of axioms, inference rules and theorems, however, forms a logical theory, which ontology languages support, but not UML [6]. The Object Constraint Language OCL is an extension of UML that adds formal specifications to UML, but the pre- and postcondition technique does not address service and process behaviour adequately. The requirements here go beyond capabilities of UML and OCL. These are requirements that are addressed with the Ontology Definition Metamodel (ODM), which is an OMG-supported activity [20]. As we will see later on, using ontology languages is actually even more suitable for the service-oriented architecture context than a UML/OCL combination due to a link between description logics (the foundation of ontology languages such as OWL) and dynamic logic (a modal logic that allows reasoning about processes).

An ontology is defined in terms of concepts and relationships. An ontology is a model of a domain made available through a vocabulary of concepts and relationships. Concrete objects of a particular concept are called instances. Relationships are used to capture properties of concepts. Properties of concepts are specified in terms of (universal or existential) quantifications over relationships with other concepts. To emphasise modelling, we often give preference to a diagrammatic, rather than textual representation of models. We present our approach here in terms of an abstract notation, usually avoiding the verbosity of XML-based representations.

3.2 CIM – Computation Independent Model

The purpose of the Computation Independent Model (CIM) is to capture a domain with its concepts and properties. Typically, two viewpoints of domain modelling can be distinguished. Concepts are represented in form of hierarchies – called the information viewpoint in MDA. Behaviour is represented in a process-based form – called the enterprise or process viewpoint in MDA, based on distributed processing concepts. We add a third aspect – the structural viewpoint – that addresses structural properties of objects and processes. Our aim is to provide a single ontological notation that can capture all three viewpoints. A process-oriented ontology shall capture two types of domain entities.

- Two types of concepts shall be distinguished: objects, which are static entities, and processes, which are dynamic entities.
- Three relationship types shall be distinguished: is a (the subclass relationship), has part (the component relationship), and depends (the dependency relationship).

Constraints, or properties, on concepts and relationships can be expressed as logical formulas. The subclass relationship is the classical form of relating concepts in ontologies. For domain-specific software systems, the composition of objects and processes from a component perspective is additional, but also essential information. Dependencies are useful to describe input-output relationships between objects and activities that process them. Specific ordering requirements on composed processes can be expressed through constraints. We will discuss the semantics of this ontology notation in Section 3.5.

We need to define or identify an ontology language that can provide the necessary notational framework. An OWL-based ontology with support for the component and dependency relationships satisfies these requirements for notational framework of our modelling approach.

Example 1 (Semantic Modelling). The example that we will use to illustrate the modelling and transformation techniques throughout the paper is taken from the banking domain. We can identify the following concepts (see Fig.1):

- objects such as account and sum (of money),
- activities such as account open, close, lodge, transfer, and enquire and processes such as for instance open; !(enquire + lodge + transfer); close which describes sequencing, iteration, and choice of activities. The principal process combinators are ';' (sequential composition), '!' (iteration), '+' (choice), and '||' (parallel composition),

Constraints, such as a precondition balance ≥ sum on the transfer activity, complement the model. The example in Fig. 1 visualises a basic domain ontology-based model for the bank account example. The information viewpoint shows a classification hierarchy (is_a) of the central objects. The structure viewpoint specifies the internal structure of both objects and processes. The process viewpoint presents the dependencies (data flow) between processes and objects. The three viewpoints are each based on a different relationship type.

Reasoning facilities of an ontological framework can be deployed to check the consistency of ontologically defined domain models. The verification of properties is important for service development and deployment due to the involvement of different clients and providers.

Example 2 (Reasoning). With instances attached to the entities, an inference engine can, for example, determine all bank account instances with a negative account balance. Another example of a reasoning task is the satisfaction of a precondition for a money transfer on a particular account.

<< ***Fig. 1. CIM-level Excerpts from a Banking Domain Ontology.*** >>

3.3 PIM – Platform Independent Model

The Platform Independent Model (PIM) shifts the focus from the computation-independent capture of the domain to a focus on architectural constraints imposed by the computational environment. Architectures and processes are the key aspects at this service modelling level. The architectural focus is on services, their architectural configuration, and interaction processes [27,28,8]. Architectural configuration addresses the interaction processes (remote invocation and service activation) between different components of a software system. Again, we will use an ontology to express these aspects.

Services are the components of the system architecture. They form the starting point of architecture modelling. Different approaches for service ontologies have been proposed. These differ in the way service and processes are represented in the ontologies – see Section 6 for a more detailed review. Since representing not only properties of services, but also their configuration and assembly into processes is important here, we use the Web Service Process

Ontology (WSPO), whose foundations were developed in [23]. This ontology focuses on the architectural perspective more than service ontology frameworks such as OWL-S [7] and WSMO [13]. OWL-S and the FLOWS ontology [31] also support service composition, but we chose WSPO as a more focussed and decidable ontology defined for service composition support. Services (and processes) in WSPO are not represented as concepts, but as relationships denoting accessibility relations between states of the system in order to realise a coherent process-oriented framework that enables modal reasoning about software behaviour.

WSPO provides a template for service and service process description. Syntactical parameter information in relation to the activities – to be implemented through service operations – and also semantic information such as pre-conditions are attached to each activity as defined in the template. This PIM service process template defines the basic structure of states and service processes. In Fig.2, the template is applied to service transfer. Instead of transfer, a composite process could also have been the central template relationship.

- Ontology concepts in this approach are states (pre- and poststates), parameters (in- and out-parameters), and conditions (pre- and postconditions).
- Two forms of relationships are provided in the ontology. The services or processes themselves are called transitional relationships. Syntactical and semantical descriptions – here parameter objects (syntax) and conditions (semantics) – are associated through descriptional relationships.

WSPO can be distinguished from traditional service ontologies by two specific properties. Firstly, based on an extension of description logics [3], it adds a relationship-based process sublanguage enabling process expressions based on iteration, sequential and parallel composition, and choice operators. Secondly, it adds data to processes in form of parameters that are introduced as constant process elements into the process sublanguage. This ontological representation in WSPO is actually an encoding of a simple dynamic logic (a logic of programs) in a description logic format [23], allowing us to avail of modal logic reasoning about processes in this framework.

<Fig. 2. WSPO Service Process Template for the PIM-layer applied to Service 'transfer'.>

Example 3a (Semantic Service Modelling). A number of individual services – such as open, balance, lodge, transfer, and close – have been defined. A semantic model of each of the services comprises syntactical and semantical aspects. In Fig. 2, these functional properties of service 'transfer' are illustrated, where the input- and output-parameter objects and the pre- and postcondition are modelled. The constraint from the CIM model, see Example 1, is here integrated as a precondition.

Example 3b (Semantic Process Modelling). The architecture- and process-oriented PIM model of the bank account focuses on the activities and how they are combined to processes. The abstract process

open; !(enquire + lodge + transfer); close

defined in Example 1 describes a sequence of account creation, an iteration of a choice of balance enquiry, lodgement, and transfer activities, and a final account closing activity. This process, which is another example of a transitional relationship in the PIM template, can be represented in WSPO as a composed relationship expression (here in textual representation):

open ° (acc);
! (enquire ° (acc); lodge ° (acc, sum); transfer ° (from, to, sum));
close ° (acc)

with parameter data attached to service names using the functional application operator °.

WSPO actually formalises our understanding of the central service and process notions in the context of service-based software systems.

Ontologies enable reasoning about specifications. WSPO enables reasoning about the composition of services in architectures. In [23], we have presented an ontological matching notion that can be applied to determine whether a service provider can be connected to a service user based on their individual service and process requirements. A classical refinement notion for operations [17] and a simulation notion for processes [29] in description logic format form the basis for this matching definition.

Example 4 (Reasoning). Assume that in order to implement an account management process, a transfer service needs to be integrated. For any given state, the process developer might require

$$\forall \text{preCond} \,.\, (\text{balance} > \text{sum})$$

and

$$\forall \text{transfer} \,.\, \forall \text{postCond} \,.\, (\text{balance}() = \text{balance}()@\text{pre} - \text{sum})$$

which would be satisfied by a provided service with

$$\forall \text{preCond} \,.\, \text{true}$$

and

$$\forall \text{transfer} \,.\, \forall \text{postCond} \,.\, (\text{balance}() = \text{balance}()@\text{pre} - \text{sum}) \wedge (\text{lastActivity} = \text{'transfer'})$$

based on a refinement condition – weakening the precondition and strengthening the postcondition. The @-construct, known from OCL, refers to the attribute in the prestate.

The refinement notion used in the example above is based on the consequence inference rule from dynamic logic and integrates the pre/postcondition technique into WSPO – which demonstrates the benefit of using the non-standard interpretation of concepts and relationships and the link to dynamic logic.

While architecture is the focus of this model layer, our approach does not qualify as an architecture description language [16], although the aim is also the separation of computation (within services) and communication (interaction processes between services). Architecture description languages usually provide notational means to describe components (here services), connectors (channels between services), and configurations (the assembly of instantiations of components and connectors). Our approach comes close to this aim by allowing services as components and process expressions as configurations to be represented.

3.4 PSM – Platform Specific Model

Our platform is the Web services platform – consisting of languages, protocols, and software tools. Models for the platform-specific layer (PSM) need to address two aspects: a platform model and implementation specific models. The platform model is here constrained by the Web services technologies and its service-oriented architecture principles. The implementation-specific models characterise the underlying models of the predominant languages of the platform. The platform in our case is different from typical MDA platforms such as Java, .NET, or CORBA where the generation of executable programs is at the centre. The Web services platform is about abstract syntactical service descriptions (WSDL), abstract

semantical service descriptions (e.g. WSMO or OWL-S), and service process definitions (e.g. WS-BPEL or WS-CDL). Transformations into this layer are therefore distinctively different from traditional PIM-to-PSM mappings. We focus on models for the WSMO and WS-BPEL platform languages here – although the ultimate aim of model-driven development is to provide transformations for a range of target languages.

Interoperability of services is a key objective of the Web services platform. Two concerns determine the techniques used at this layer: the abstract description of services to support their discovery and remote invocation and the standardised assembly of services to processes. Two different models supporting executable and tool-supported languages are therefore relevant here:

- Description and Discovery. Abstract syntactical and semantical service interfaces shall be supported. The Web Services Description Language (WSDL) supports syntactical information needed for service invocation. We, however, focus here on semantically enhanced descriptions enabled. Services as the basic components of processes can be represented as concepts in ontologies [25]. This approach is followed by widely used service ontology frameworks such as OWL-S [7] and WSMO [13].
- Processes and Composition. The Business Process Execution Language for Web Services (WS-BPEL) is one of the proposed service coordination languages [26,33]. WS-BPEL specifications can be created by converting process expressions from WSPO.

The benefit of using an ontology for description and discovery can easily be seen when the discovery and matching of OWL-S or WSMO-based semantic service descriptions of a range of functional and non-functional properties is compared with syntax-oriented WSDL descriptions.

Example 5a (Semantic Service Description). WSMO descriptions capture syntactical and semantical descriptions as WSPO does, see Examples 3 and 4. It adds, however, various non-functional aspects that can be included into the discovery and matching task. WSMO defines a template for the representation of service-related knowledge, see Fig. 3. The WSMO concepts are the central services concept and auxiliary domains for descriptional entities, i.e. expressions of different kinds. Relationships in the template represent service properties of two kinds. Properties such as preCond, postCond, assumption, and effects relating to the service semantics are called capabilities. Properties such as messageExchange are syntactically oriented interface aspects.

<<***Fig. 3. Ontological Service Template (WSMO) with Interface and Capability Aspects.***>>

Standardised description and invocation formats enable interoperability. Required functionality for a particular process can be retrieved from other locations. An example is an authentication feature for an online banking system. The authentication service, integrated into the banking process, can be provided at a remote location by a third-part provider.

Example 5b (Service Process Definition). WS-BPEL offers a range of control flow operators including sequence, flow (parallel composition), switch (choice), and while (iteration). These are direct counterparts of the WSPO relationship combinatory. The result of the transformation for the account management process can be seen in Appendix 1.

3.5 Semantics of the Ontology Layers

Ontology languages are logics defined by interpretations and satisfaction relations on semantical structures such as algebras (sets and relations) and state-based labelled transitions systems (e.g. Kripke transition systems). We can exploit the description logic foundation of ontology languages such as OWL [3]. While a full treatment is beyond the scope of this paper,

we address the central ideas since the definition of ontology transformations requires underlying formal semantical models. A semantical metamodel for each of the layers can be formulated based on standard approaches in this context:

- A domain ontology – the CIM layer – can be defined in terms of sets (for concepts) and relations (for relationships). We have proposed OWL-DL, which is defined in terms of standard description logic [3].
- The architectural and process aspects – the PIM layer – can be defined in terms of labelled transition systems, such as Kripke transition systems, where sets represent states and relations represent transitions between states. WSPO is also defined in terms of description logics with some extensions that exploit a link to dynamic logic [3,30].
- The interoperability aspects – s the PSM layer – can be split into interface (defined in terms of sets and relations) and configuration and process behaviour (defined in terms of state transition mechanisms). The proposed service description notation WSMO is also rigorously defined in terms of logics. BPEL is a workflow and business process language whose central concepts can be defined in terms of a process calculus along the lines of the process expression sublanguage of WSPO [15,22].

In the future, these frameworks can be mapped onto the soon to be standardised OMG-supported Ontology Definition Metamodel (ODM). This can be expected to be straightforward due to an ODM-OWL mapping as part of ODM.

4 Ontology-based Model Transformations

Without explicitly defined transformations, a layered modelling approach will not be feasible. Transformations between the model layers need to be automated to provide required tool support and to enable the success of the approach. Following the OMG-style for MDA transformations, we define transformation rules based on patterns and templates. While it is evident that the transformations we require here are about adding new structures, for instance notions of state and state transition for the architectural PIM layer, the original model should be recoverable and additional application information on that layer should not be added. What we aim at is therefore not a refinement or simulation notion in the classical sense – although these notions will help us to define the transformations.

The main aim of transformations in traditional MDA is full automation, which is not our central objective here. Supporting and guiding the software architect, however, is important – see Fig. 4 for an overview of the transformation approach.

- The CIM-to-PIM mapping changes the focus from domain modelling to architecture modelling, which might require some additional information. However, given a detailed domain model addressing the three viewpoints, all information required by the PIM template is available.
- The PIM-to-PSM mapping requires additional information, in particular for the comprehensive abstract description of functional and non-functional aspects.

Both transformations can in any case automatically generate structured templates and skeletons that contain core elements.

<< ***Fig. 4. Mappings between the Ontology Layers – Overview.*** >>

4.1 CIM-to-PIM Mapping

The CIM-layer supports abstract, computation-independent domain modelling. This model is mapped to a computation-oriented, but still platform-independent service-based model. The PIM-layer supports analysis and reasoning for architecture and process aspects, such as

configuration and composition, on an abstract level. Consequently, information only needs to be added to a CIM to provide a sufficient level of structure for the PIM-level if the process viewpoint is not adequately modelled. A process-specific PIM template, see Fig. 3 for a template application to the banking context, guides the transformation process. We have defined the rules for the CIM-to-PIM transformation in Table 1.

Table 1. Transformation Rules for the CIM-to-PIM Mapping.

Rule	Aspect	Description
CP0	template	For each process element in the CIM, create a PIM template.
CP1	process element	The PIM process element is the process element of CIM.
CP2	states	Create default concepts for pre- and post-states.
CP3	syntax	For each in- and out-parameter of processes, create a separate syntax (object) element.
CP4	semantics	Create pre- and postconditions depending on availability of external additional information in form of constraints.
CP5	process	If process expressions available in form of constraints, then expressions create complex process using relationship expressions in WSPO.

In MDA, the transformation steps are defined in terms of model markings and applications of templates. Marks are annotations (or metadata) of entities in the original model to support the mapping that indicates how these entities are used in the target model. Marks can support the determination of the mapping template to be deployed. The CIM-to-PIM transformation rule CP0, which defines the creation of a PIM-template for CIM-concepts marked as 'process', is an example of this.

Example 6 (CIM-to-PIM Transformation). Fig. 2 represents the result of the transformation of the transfer process from Fig. 1 using the rules defined in Table 1. The transfer concept in Fig. 1 is marked as a process, which based on rule CP0 creates a PIM process template with explicit states (rule CP2). The CIM concept transfer becomes the transitional relationship element at the centre of the PIM template (rule CP1). The input and output elements, associated to transfer using dependencies (see Fig. 1), are mapped to syntax descriptions (rule CP3). Equally, additional constraints in the CIM such as pre- and postconditions are mapped to the PIM semantical descriptions (rule CP4).

A detailed CIM with constraints actually contains all information needed to fill the WSPO template. In general, not all CIM information is used. For instance, structural aspects are only relevant for the platform specific layer.

4.2 PIM-to-PSM Mapping

The platform-specific model (PSM) is defined in our approach by two separate models: service metadata based on ontology descriptions to address service discovery, and process orchestration and choreography descriptions to address service composition. The corresponding transformation rules for these two aspects – we chose WSMO for ontology-based description and WS-BPEL for service orchestration to illustrate this mapping – are presented in Table 2.

Table 2. Transformation Rules for the PIM-to-PSM Mapping.

Rule	Aspect	Description
PP1	WSMO	From the WSPO-based PIM, map process relationships to WSMO service concept and fill messageExchange and

		pre/postCond properties accordingly, see WSMO-template in Fig. 4.
PP1.1	WSMO messageExchange	Map the WSPO in and out objects onto WSMO message exchange descriptions.
PP1.2	WSMO pre-/postconditions	Map the WSPO pre- and postconditions onto WSMO pre and postconditions.
PP2	WS-BPEL	The complex WSPO process relationships can be mapped to BPEL processes.
PP2.1	WS-BPEL process partners	For each process create a BPEL partner process
PP2.2	WS-BPEL orchestration	Convert each process expression into BPEL-invoke activities and the client side BPEL-receive and -reply activities at the server side.
PP2.3	WS-BPEL process activities	Convert the process combinators ';', '+', ' !', and '||' to the BPEL combinators sequence, pick, while, and flow, resp.

The WSPO-to-WSMO mapping copies functional properties – both syntax and semantics – to the PSM. Similar to states that are added to CIMs to provide the structure to express process behaviour, we add structure in form of non-functional aspects to PIMs to support further descriptions for service discovery. Due to the nature of the platform requiring abstract service descriptions, the aspect is on the same level of abstraction as the platform-independent model.

Example 7 (PIM-to-PSM Transformations). The WSMO example in Fig. 3 is the result of mapping the PIM, presented in Fig. 2, to the Web services platform layer according to rule PP1 defined in Table 2. Syntactical elements for the interface and semantical capabilities such as pre- and postconditions are directly mapped from the corresponding WSPO elements according to the transformation rules PP1.1 and PP1.2.

The WSPO-to-WS-BPEL mapping converts process expressions into a BPEL business process skeleton, see Fig. 4. WS-BPEL is an implementation language for process execution in form of process orchestrations. WS-BPEL implementations are supported by service engines available from various providers. Since WSPO comprises a process expression sublanguage similar to WS-BPEL, a WSPO model can be fully translated into WS-BPEL. In order to form a complete, executable WS-BPEL specification, a number of additional elements have to be specified, which includes namespace and partner type information. This is platform-specific information and is therefore not included in WSPO.

A central benefit of MDA is the provision of several transformations for a given CIM to support different platforms. For instance, we could have provided transformations for OWL-S and WS-CDL as alternatives to WSMO and WS-BPEL, respectively, allowing user to switch between platform languages easily. These automated transformations would also allow circumventing implementation restrictions by providing a richer set of process combinators at the PIM level, supported by transformations onto a combination of simpler combinators.

4.3 Formal Mapping Definitions

Our focus here is the illustration of the different modelling capabilities of ontology languages and ontologies on the different model layers. Our objective is to motivate the need for and the benefits of a layered ontological modelling and transformation approach. A formal model of transformations is beyond the scope of this paper. Languages such as QVT [21], like ODM also supported by the OMG, can provide standardised frameworks in the future. Graph transformation and graph grammars provide suitable formal frameworks to formalise the transformation rules [12,4]. We have used graphs as the visualisation mechanism for ontological models. Graph-based models and CIM-to-PIM transformation semantics are

therefore a natural combination. The semantics of a CIM can be seen as a directed labelled graph with nodes (objects and processes) and edges (relationships). The semantics of a PIM can be seen directed labelled graph, where descriptional and transitional roles are distinguished. This is equivalent to a Kripke transition system, the semantic structure underlying description logic specifications (see Section 3.4). This can be implemented as a graph expansion, where essentially state concepts are introduced. The original CIM can be retrieved by projecting on individual PIMs and then merging all process PIMs into one CIM. Formal transformation definitions are required to establish the correctness of the transformation in terms of semantics preservation.

5 Modelling – Standards and Interoperability

Interoperability and model integration is a central issue in model-driven development. The remodelling of existing UML-models in ontology format cannot be expected from a software developer. Automated conversion between the formats is consequently needed. Although UML and ontology languages are not the same, they do overlap substantially and therefore allow the conversion between UML models and OWL-based ontologies. A common metamodel with mappings to UML and OWL can solve this problem.

A standardised ontology metamodel like ODM [20] allows us to integrate our technique further with existing standards. ODM provides mappings between ontology and other modelling languages. A UML profile for ontologies makes UML's graphical notation available. MOF compliancy for ODM facilitates tool support. XMI, i.e. production rules using XSLT, can be used to export model representations to XML, e.g. to generate XML Schemas from models using the production rules. We have summarised the MDA framework and compared it with our proposed extension in Fig. 5. Our framework is shown on the left-hand side and the OMG MDA-context on the right-hand side. It illustrates the interoperability and integrability of our ontology models with UML-based models.

<< ***Fig. 5. Overview of MDA and Ontology-based Service Modelling (with transformations between the layers and the influence of ODM for the ontology layers).*** >>

In addition to the standards relating to modelling and description, transformation standards also need some attention. Declarative query and transformation languages have recently been promoted to replace older, procedural languages such as XSLT. The Query View Transformation language QVT is an example [21]. We have used an ad-hoc approach in our implementation based on a format supported by the OMG in the MDA framework. In the future, once QVT is widely supported, it is another option to transformation specification and implementation.

6 Related Work

Service ontologies are ontologies to describe Web services, essentially to support their semantics-based discovery in Web service registries. WSMO [13] and OWL-S [7] are the two predominant examples that have been developed and used extensively in the recent past. WSMO is not an ontology, as OWL-S is, but rather a framework in which ontologies can be created. The Web Service Process Ontology WSPO [23] is also a service ontology, but the focus has shifted here to the support of description of and reasoning about service composition and service-based architectural configuration. Both OWL-S and WSPO are or can be written in OWL-DL. WSMO is similar to our endeavour here, since it is a framework of what can be seen as layered ontology descriptions. We have already looked at the technical

aspects of WSMO descriptions. WSMO supports the description of services in terms more abstract assumptions and goals and more concrete pre- and postconditions. The FLOWS ontology from the Semantic Web Services Framework is a recent service process ontology very similar to WSPO [32]. Although FLOWS is a more expressive modelling framework, WSPO is in contrast to FLOWS decidable [23], which adds to the tractability of our solution.

In addition to service description, service modelling as part of a service engineering approach is a staged, layered process. We have already discussed the OMG efforts to develop an ontology definition metamodel (ODM) for layered model-driven architecting in the previous section, which, due to its support of OWL, allows integration with UML-style modelling. ODM, however, is a standard addressing ontology description. The reasoning component, which is important here, would need to be addressed in more detail. Some developments have started exploiting the connection between OWL and MDA. In [9], an MDA-based ontology architecture is defined, which includes aspects of an ontology metamodel and a UML profile for ontologies – corresponding to ODM. A transformation of the UML ontology to OWL is implemented. The works by Djurić et.al. [9,10] and the OMG [19,20], however, need to be carried further to address the ontology-based modelling and reasoning of service-based architectures.

In particular, the Web services platform needs to be addressed in the context of Web-based ontology technology. Grønmo et.al. [11] introduce – based on ideas from [9] – an approach similar to ours. Starting with a UML profile based on activity diagrams, services are modelled. These models are then translated into OWL-S. Although the paper discusses process composition, this aspect is not detailed. We have built on their work in this respect by considering process compositions and by mapping into a service ontology that focuses on providing explicit support for service processes. Other authors [15] have directly connected UML modelling with WS-BPEL code generation, without the explicit ontology framework. Integrating ontologies, however, enhances the semantic modelling and reasoning capabilities in the context of service architectures. These approaches go beyond our framework in that UML-style graphical modelling is provided. We have discussed in Section 5 how this could be introduced into our solution

7 Conclusions

The development of service-based software architectures requires the integration of domain modelling and architectural configuration aspects in order to implement services as reusable and composable entities in a process-oriented environment. We have presented an integrated, layered semantic service modelling and transformation framework. We have demonstrated that different ontology-driven modelling techniques exist to support these different activities. The effort leading towards model-driven architecture acknowledges the importance of modelling for the architectural design of software systems:

- Ontologies are a natural choice to enhance modelling capabilities. While this is recognised in the community, we have exploited the new degree of sharing and ubiquity enabled through Web ontology languages and the reasoning capabilities of logic-based ontology languages for service engineering.
- Ontology-based transformations allow the seamless and coherent transition from one development focus to another. These ontology transformations allow the integration of domain modelling, architectural design, and the description and discovery of services.

Our approach addresses a software service-specific solution, reflecting the current development of the Web services and the Semantic Web. The primary platform we aim to support is service-oriented architecture with the second Web services generation focusing on processes and composition, utilising the Semantic Web with its ontology technology support.

A platform of the expected importance in the future, such as the Web services platform, requires an adequate and platform-specific MDA-based service engineering solution. Service-specific solutions, which have only started to emerge due to the novelty of the platform, are required since the platform with service and process execution and publishable service description is different from the classical MDA focus on component and object platforms. Our framework provides effective support for the software architect through service architecture modelling on an abstract level. The software architects benefits from semantic modelling and reasoning, improved maintainability, and automated generation of potentially a range of different platform specific implementations. In addition to the banking example, which we have implemented to study distribution patterns of services, we have also investigated service-based learning technology systems implementations. Although not all aspects are currently fully automated, as our transformation discussion shows, both areas have demonstrated the benefits of model-driven development in terms of improved software change and maintenance through abstraction and reasoning capabilities.

A critical problem that has emerged from this investigation is the need for conformity and interoperability. As MDA and the Web as a platform are developed and standardised by different organisations, this can potentially cause problems. The current developments, such as the Ontology Definition Metamodel (ODM), however, aim to reconciliate some of these problems. With ODM, our proposed ontologies can, due to their grounding in OWL-DL, be integrated into the ODM. This enables interoperability between ontology-based and traditional models.

Our aim was to demonstrate the benefits and the feasibility of layered ontology-based semantic modelling and transformation for service-oriented architecture. We have developed a semantic modelling and transformation framework. While we have developed reasoning support specific to architectural modelling activities such as refinement- and simulation-based matching, more techniques are possible that exploit the full range of modal reasoning for service description, discovery, and composition and architectural configuration. Reasoning about safety and liveness conditions can enhance the semantic modelling potential further.

References

[1] R. Allen, D. Garlan, A Formal Basis for Architectural Connection, ACM Transactions on Software Engineering and Methodology, 6 (1997) (3), pp. 213–249.

[2] G. Alonso, F. Casati, H. Kuno, V. Machiraju, Web Services – Concepts, Architectures and Applications, Springer-Verlag, Berlin, 2004.

[3] F. Baader, D. McGuiness, D. Nardi, P.P. Schneider (editors), The Description Logic Handbook, Cambridge University Press, Cambridge, 2003.

[4] L. Baresi, R. Heckel, Tutorial Introduction of Graph Transformation: A Software Engineering Perspective, In A. Corradini, H. Ehrig, H.-J. Kreowski, and G. Rozenberg (editors), Proc. 1st Int. Conference on Graph Transformation ICGT02, Springer-Verlag, LNCS 2505, 2002, pp. 402-429.

[5] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice (2nd Edition), SEI Series in Software Engineering, Addison-Wesley, Boston, 2003.

[6] M.D. Daconta, L.J. Obrst, K.T. Smith, The Semantic Web, Wiley, Indianapolis, 2003.

[7] DAML-S Coalition. DAML-S: Web Services Description for the Semantic Web, In I. Horrocks and J. Hendler (editors), Proc. First International Semantic Web Conference ISWC 2002, Springer-Verlag , LNCS 2342, 2002, pp. 279–291.

[8] N. Desai, M. Singh, Protocol-Based Business Process Modeling and Enactment, Proc. International Conference on Web Services ICWS 2004, IEEE Press, 2004, pp. 124–133.

[9] D. Djurić, MDA-based Ontology Infrastructure, Computer Science and Information Systems (ComSIS), 1 (2004) (1), pp. 91–116.

[10] D. Gašević, V. Devedžić, D. Djurić, MDA Standards for Ontology Development – Tutorial, In International Conference on Web Engineering ICWE2004, 2004.

[11] R. Grønmo, M.C. Jaeger, H. Hoff. Transformations between UML and OWL-S, In A. Hartman and D. Kreische (editors), Proc. Model-Driven Architecture - Foundations and Applications, Springer-Verlag, LNCS 3748, 2005, pp. 269-283.

[12] J. Kong, K. Zhang, J. Dong, G. Song, A Graph Grammar Approach to Software Architecture Verification and Transformation, Proc. 27th Annual International Computer Software and Applications Conference COMPSAC'03, 2003, pp. 492-497.

[13] R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, D. Fensel, Web Service Modeling Ontology, Applied Ontology, 1 (2005) (1), pp. 77-106.

[14] D.J. Mandell, S.A. McIllraith, Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation, In D. Fensel, K.P. Sycara, and J. Mylopoulos (editors), Proc. International Semantic Web Conference ISWC'2003, Springer-Verlag, LNCS 2870, 2003, pp. 227–226.

[15] K. Mantell, From UML to BPEL - Model Driven Architecture in a Web services world, IBM, http://www-128.ibm.com/developerworks/webservices/library/ws-uml2bpel/, 2005, (visited 31/07/2006).

[16] N. Medvidovic, R.N. Taylor, A Classification and Comparison framework for Software Architecture Description Languages, Proceedings European Conference on Software Engineering / International Symposium on Foundations of Software Engineering ESEC/FSE'97, Springer-Verlag, 1997, pp. 60–76.

[17] C. Morgan, Programming from Specifications, Addison-Wesley, London, 1994.

[18] E. Newcomer, G. Lomow, Understanding SOA with Web Services, Addison-Wesley, Boston, 2005.

[19] Object Management Group, Model-Driven Architecture MDA Guide V1.0.1, OMG, 2003.

[20] Object Management Group, Ontology Definition Metamodel (OMG RFP: as/2003-03-40), OMG, 2003.

[21] Object Management Group, MOF QVT Final Adopted Specification (OMG RFP: ptc/05-11-01), OMG, 2005.

[22] C. Pahl, Layered Ontological Modelling for Web Service-oriented Model-Driven Architecture, Proc. European Conference on Model-Driven Architecture ECMDA'2005, Springer-Verlag, LNCS 3748, 2005, pp. 88-102.

[23] C. Pahl, An Ontology for Software Component Matching. International Journal on Software Tools for Technology Transfer, Special Edition on Component-based Systems Engineering, 7 (2006), pp. 1-10.

[24] M. Paolucci, T. Kawamura, T.R. Payne, K. Sycara, Semantic Matching of Web Services Capabilities, In I. Horrocks and J. Hendler (editors), Proc. First International Semantic Web Conference ISWC 2002, Springer-Verlag, LNCS 2342, 2002, pp. 279–291.

[25] T.R. Payne, O. Lassila, Semantic Web Services, IEEE Intelligent Systems, 19 (2004) (4), pp. 14-15.

[26] C. Peltz, Web Service orchestration and choreography: a look at WSCI and BPEL4WS, Web Services Journal, 3 (2003) (7).

[27] F. Plasil, S. Visnovsky, Behavior Protocols for Software Components, ACM Transactions on Software Engineering, 28 (2002) (11), pp. 1056–1075.

[28] J. Rao, P. Küngas, M. Matskin, Logic-Based Web Services Composition: From Service Description to Process Model, Proc. International Conference on Web Services ICWS 2004, IEEE Press, 2004, pp. 446–453.

[29] D. Sangiorgi, D. Walker, The π-calculus - A Theory of Mobile Processes, Cambridge University Press, Cambridge, 2001.

[30] K.A. Schild, Correspondence Theory for Terminological Logics: Preliminary Report, Proceedings 12th International Joint Conference on Artificial Intelligence, 1991, pp. 466-471.

[31] Semantic Web Services Language (SWSL) Committee, Semantic Web Services Framework (SWSF), http://www.daml.org/services/swsf/1.0/, 2006, (visited 31/07/2006).

[32] World Wide Web Consortium, Web Services Architecture Definition Document, http://www.w3.org/TR/ws-arch, 2006, (visited 31/07/2006).

[33] WS-BPEL Coalition, WS-BPEL Business Process Execution Language for Web Services – Specification Version 1.1, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel, 2004, (visited 31/07/2006).

Appendix 1. WS-BPEL Code

```
<?xml version="1.0" encoding="utf-8"?>
<process name="AccountProcess"
      ...                                  <!– namespaces -->
   <partnerLinks>
      <partnerLink      name="client"           ... />
      <partnerLink      name="loginServer"      ... />
      <partnerLink      name="accountManager"   ... />
   </partnerLinks>
   <variables>
      <variable   name="userID"
                  messageType="ID" />
      ...
   </variables>
   <sequence>
      <receive    partnerLink="client"    <!-- login -->
                  portType="ClientProcess"
                  operation="login"
                  variable ="userID“
                  variable="userPWD" />
      <invoke     partnerLink="loginServer"
                  portType="loginSrv"
                  operation="login"
                  inputVariable="userID"
                  inputVariable="userPWD" />
      <while>
          <switch>
            <case condition=""            <!-- balance enquiry -->
               <invoke  partnerLink="accountManager"
                  portType="accMngr"
                  operation="enquire"
                  inputVariable="account"
                  outputVariable="balance" />
               <receive partnerLink="accountManager"
                  portType="accMngr"
                  operation="enquire-res"
                  variable="balance" />
            </case>
            <case condition=""            <!-- transfer-->
               <invoke  partnerLink="accountManager"
                  portType="accMngr"
                  operation="transfer"
                  inputVariable="account"
                  inputVariable="sum" />
               <receive partnerLink="accountManager"
                  portType="accMngr"
                  operation="transfer"
                  variable="balance" />
            </case>
            <case condition=""            <!-- lodge-->
               <invoke  partnerLink="accountManager"
                  portType="accMngr"
                  operation="lodge"
                  inputVariable="account"
                  inputVariable="sum" />
               <receive partnerLink="accountManager"
                  portType="accMngr"
                  operation="lodge"
                  variable="balance" />
```

```
            </case>
          </switch>
      </while>
      <receive    partnerLink="client"    <!-- logout -->
                  portType="ClientProcess"
                  operation="logout"
                  variable="userID " />
      <invoke     partnerLink="loginServer"
                  portType="loginSrv"
                  operation="logout"
                  inputVariable="userID" />
   </sequence>
</process>
```

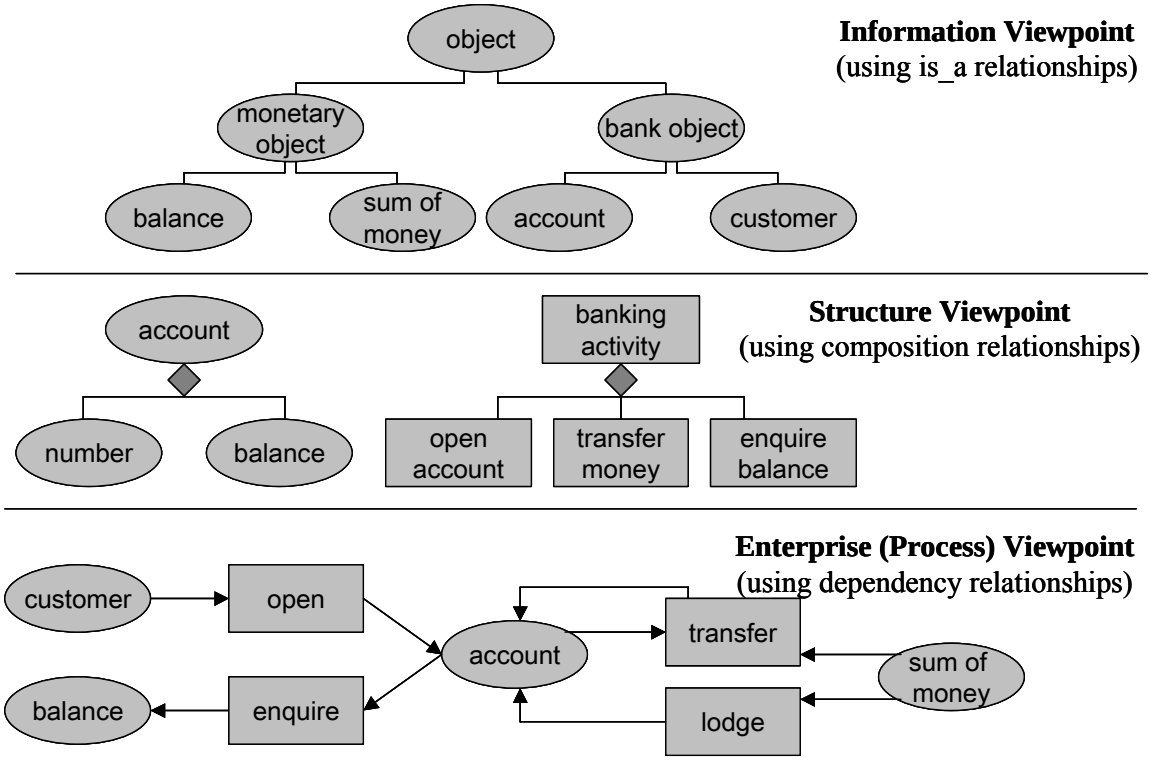

Fig. 1. CIM-level Excerpts from a Banking Domain Ontology.

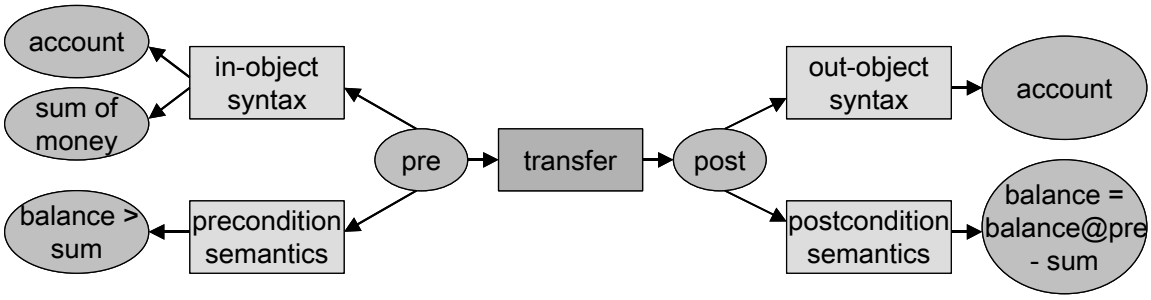

Fig. 2. WSPO Service Process Template for the PIM-layer applied to Service ‘transfer’.

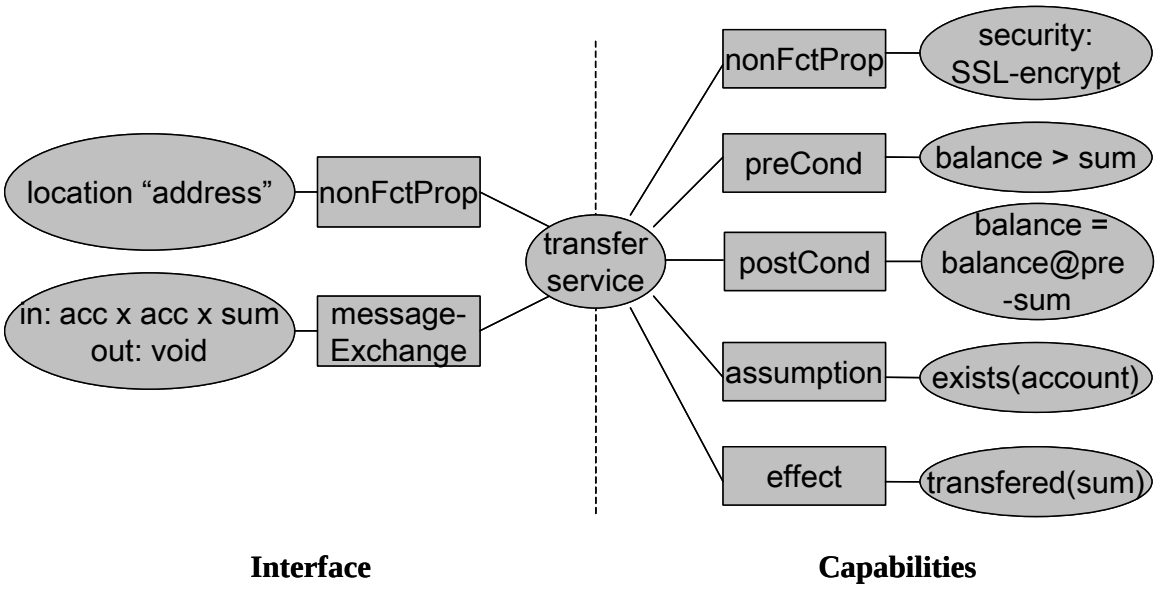

Fig. 3. Ontological Service Template (WSMO) with Interface and Capability Aspects.

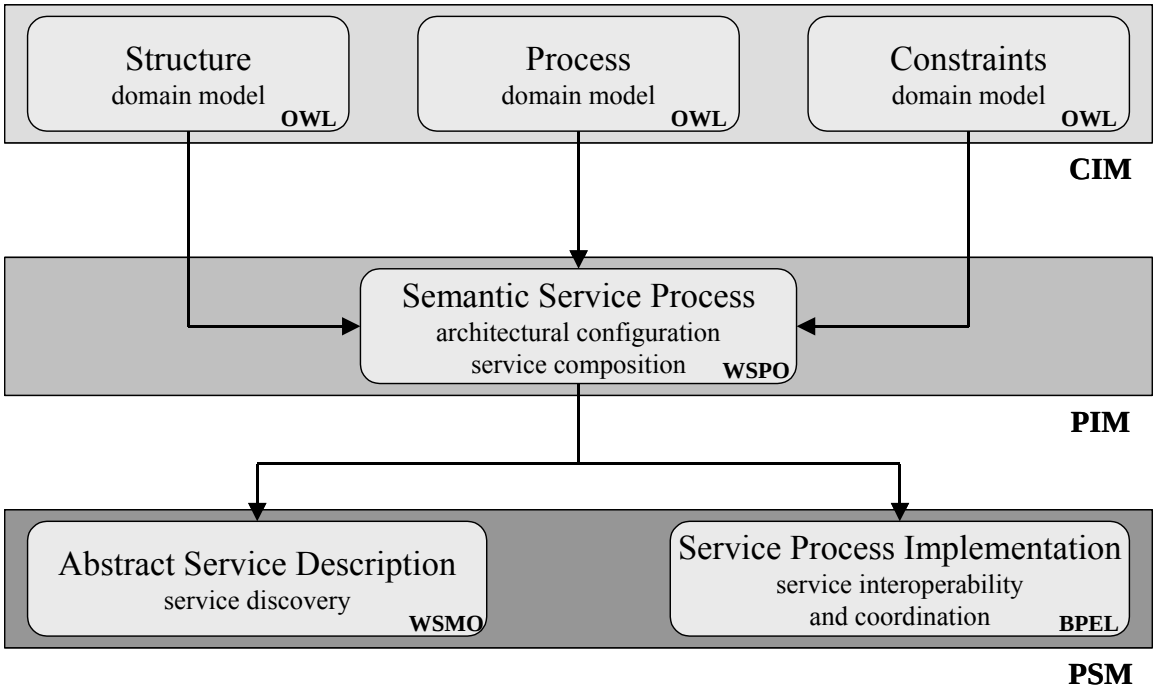

Fig. 4. Mappings between the Ontology Layers – Overview.

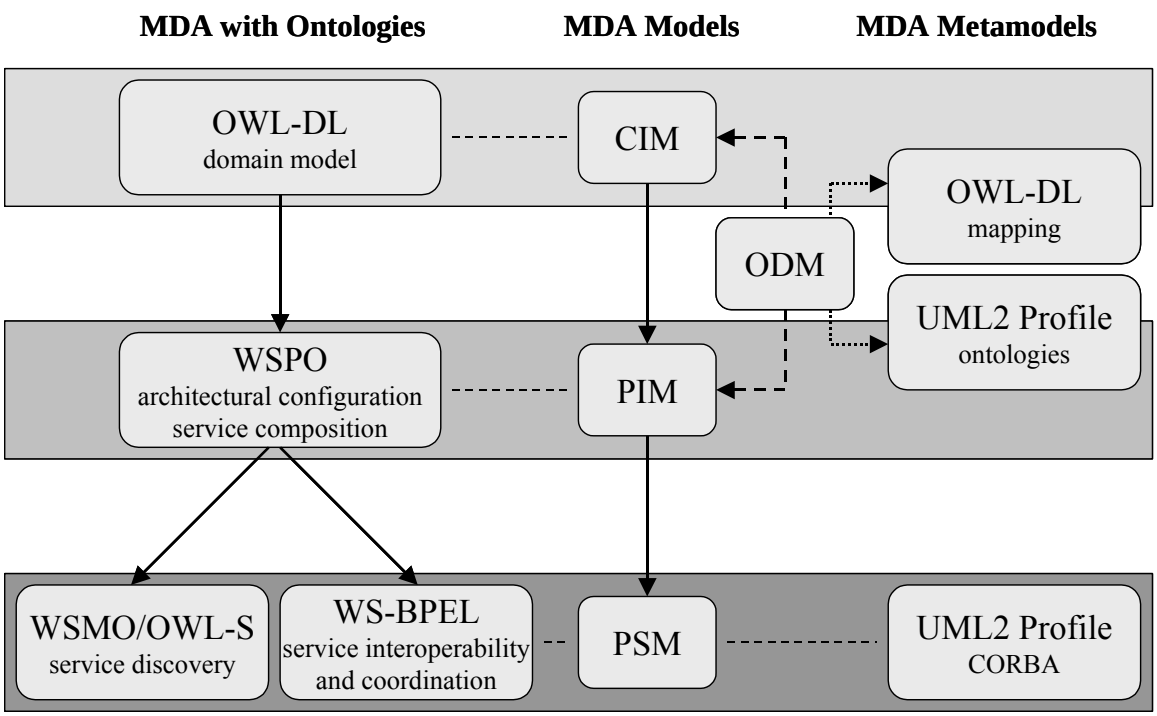

Fig. 5. Overview of MDA and Ontology-based Service Modelling (with transformations between the layers and the influence of ODM for the ontology layers).