An Ontology-Driven Architecture for Semantic Web Service Composition

Carlos Granell¹, Dolores María Llidó¹, Rafael Berlanga¹, and Michael Gould¹

¹ Department of Information Systems
Universitat Jaume I, Castellón, Spain
{carlos.granell, doloresmaria.llido, berlanga, gould}@uji.es

Abstract. Nowadays Model-Driven Architecture (MDA) techniques applied to Web service development are considered an emerging trend in web engineering. MDA emphasizes abstract modeling, considering models at three different levels of abstraction, namely Computer-Independent Model (CIM), Platform-Independent Model (PIM) and Platform-Specific Model (PSM).

Typically, MDA process follows a top-bottom approach in which initial PIM models are transformed into PSM models until reaching the required implementation. However, in our opinion, MDA process lacks the ability to reuse existing PIM, and especially CIM models to be integrated in new web applications. In this paper, we present an ongoing ontology-driven architecture that enables using Semantic Web mechanisms in a MDA-based framework to facilitate the reuse of both semantic web service composition and knowledge.

Keywords: Ontology-driven architecture, semantic web services, model-driven development, abstract patterns, knowledge reuse.

1 Introduction

The Model Driven Architecture (MDA) [11] has recently become one of the most emerging techniques which are applied to solve the problem of Web service composition. In short, MDA defines three level of abstraction or viewpoints from which a service composition can be represented: CIM, PIM and PSM. However, most MDA-based approaches for web service composition are focused mainly in the PIM and PSM levels, following a top-bottom approach in which initial PIM models are transformed into PSM models until reaching the required implementation. However, such approaches ignore the advantages of reusing CIM models. For example, the same CIM model can be applied to different contexts, though the generated set of PIMs requires different configurations according to each context features.

In this paper, we address the lack of reuse of services and knowledge presenting an ongoing ontology-driven architecture that provides a CIM’s Knowledge Base capable of reasoning –classifying, composing, and querying– over services in an abstract way. Also we describe a querying-driven process at CIM level that achieves structured
semantic answers given user goals or queries, underlying the importance of CIM for improving the reuse of services and knowledge between CIM and PIM levels.

The paper is structured as follows. Next section introduces background concepts. Section 3 describes the proposed ontology-driven architecture. Section 4 and 5 are devoted to the CIM level, describing the structure of the knowledge base and the query processing. Finally, Section 6 concludes the paper.

2 Background and Related Work

Most research efforts in web service composition realm are devoted to the PIM and PSM models [5][7][8]. Platform-neutral descriptions for web services are used for representing PIM models because they are not aware for example of the syntax of the service inputs and outputs. UML is an example at this level. From such PIM models, it is relatively easy to perform transformations to obtain executable service compositions at PSM level. In practice, PIM and PSM models have received the most attention to date ignoring the potential benefits offered by CIM models, identified commonly as domain models. It seems obvious that reasoning, composing, and classifying at CIM level make easier for users to reuse and understand such models rather than performing such tasks at PIM level. Discovering and reusing CIM models implies that the same way to solve semantically a problem may be applied to different contexts. For each context, a specific PIM is generated but based on the same CIM model, permitting both reusing service and knowledge [4].

OWL-S [9] aims to describe semantically Web services in machine-processable forms by using three different ontologies: Service Profile, Service Process and Service Grounding. The Service Profile defines some non-functional properties of the service and exposes some functionality (both advertisements and requests) by referencing Inputs, Outputs, Preconditions and Effects (IOPEs). The Service Process describes the behaviour of the Service in terms of processes and control constructs. The Service Grounding binds processes inputs and outputs in the Service Process to concrete implementation such as WSDL documents.

In our approach, we use OWL-S to describe services for several reasons. First, OWL-S is becoming a de facto standard for specifying Semantic Web services. Also, as we see in Section 4, the CIM knowledge base is described in Description Logic (DL) [3]. OWL introduces the OWL DL as one of the three expressive sublanguages that it exposes. OWL DL enables maximal expressiveness and guarantees computational completeness [9], making it suitable for reasoning tasks. In addition, DL provides the basis for both OWL DL and OWL-S, facilitating interoperability among such languages.

We use jointly OWL DL and OWL-S at different level of abstraction. At CIM level we create DL-semantic descriptions based on OWL DL, which in turn can be easily compared to OWL-S ontologies. As Profile and Process ontologies are independent of the platform, they can be used in the PIM Model. Profile ontology serves to specify service IOPEs whereas Process ontology is closely related to the
abstract patterns (see Section 4). Finally, at PSM level, we need the Grounding ontology, in addition to other standards such as WSDL and WS-BPEL.

3 Ontology-driven Architecture

Figure 1 illustrates the ongoing ontology-driven architecture that enables using Semantic Web mechanisms in conjunction with MDA techniques to facilitate both semantic web service composition and reuse. We first propose the use of DL-semantic descriptions to deal with two key aspects of web services: the classification of services (with respect to their IOPEs) and the classification of abstract patterns that can be applied over them. The former enables reason and compose properly candidate semantic service descriptions (concepts) which may form part of the target solution, whereas the latter allows us to validate possible abstract patterns that satisfy the selected service concepts. In our approach, both elements that are fully expressed as logic-based ontologies (with OWL-DL expressivity) constitute the semantic required to build the CIM’s Knowledge Base (see Section 4).

At CIM level, user goals are specified in terms of semantic descriptions and abstract patterns. In this way, by applying jointly reasoning techniques and the Graph Handler component we are able to define a semantic pattern as a structured composition of concepts that satisfies the requirements of user goals.

Next, the semantic pattern is validated and completed (i.e. all implicit knowledge must become explicit) using semantic matching techniques to find possible services at the PIM base. The PIM base contains OWL-S Profile ontologies, which describe semantically service IOPEs. A Semantic Composition Graph (SCG) stems from completing successfully the semantic pattern. A SCG [10] relates ontology concepts with semantic service descriptions, and it is built by matching concepts in the semantic pattern with service concepts described as IOPEs (OWL-S Profile) within the PIM base. With a SCG we can specify semantic queries and their possible execution plans, allowing us to validate if there exist any execution plan that solves the user goals. Once validated the SCG, a MOF-compliant model (labeled as Goal PIM in Figure 1) is generated from the SCG by applying specific PIM transformations. Furthermore, the MOF-compliant model expresses the process behavior of the PIM model.

At this stage, we are able to build the PSM related with the Goal PIM following traditional MDA transformations. The first transformation (PIM-PSM) permits to generate the OWL-S Grounding associated with the OWL-S Profile and Process created earlier. As grounding specifies the details of how to access concrete services, it is then necessary accessing to web services repositories. Finally, the second transformation (PSM-PSM) defines an executable WS-BPEL process ready to be executed by compliant workflow engines. Efforts in this step have been already carried out [6]. It is worth mentioning that if the resulting PSM is executed successfully, then the corresponding models at different levels are registered for future use. So, the semantic pattern (knowledge) is included in the CIM’s
DL-Knowledge Base, whereas the PIM model (OWL-S Profile) updates the PIMs Base.

This architecture is being implemented and applied to emergency management scenario. In this scenario, a new PIM is usually required when a new emergency arise, and it should be designed and deployed as quick as possible. Thus, the reuse of already built models (at different level of abstraction) can be crucial. In the current implementation, final PSM models are WSBPEL processes over existing (composite) web services specially designed for this scenario.

Fig. 1. Ontology-driven architecture for semantic Web service composition and reuse.

4 CIM’s Knowledge Base

This section focuses on how the CIM’s Knowledge Base (KB) is structured and how query processing and knowledge reuse are achieved. The rest of this section describes in detail the elements involved in the CIM and their relationships.

The KB comprises the underlying domain ontology (e.g. geospatial ontology), the semantic descriptions of the available services, and the semantics of the composition abstract patterns. We assume that the KB is expressed in DL [3], which provides the basis for OWL DL and OWL-S [9]. Therefore, the KB can be exported-imported
through the OWL-DL language. The KB provides a reasoner that classifies concepts and services, check for composition consistency and provide answers to user requests.

We use a single ontology, which provides a common vocabulary for specifying the semantics of all services and queries, in contrast to a multiple ontology approach in which each service is described by a different ontology. The latter approach presents the problem of comparing ontologies because of the lack of a common vocabulary. Notice that a global ontology may produce different domain views, which can be shared by services applied to the same problem. In our approach, using domain views at CIM level facilitates reasoning on the concepts representing services and queries.

In order to simplify as much as possible the semantic description of services, we only define two concepts, Service and Profile, and the following properties for their characterization: hasProfile, hasInput, hasOutput, hasBLogic, hasPreCondition and hasEffect. The former defines the semantics of the services according to their inputs (hasInput) and outputs (hasOutput). The property hasBLogic expresses the relation between each service and the state variables it requires (hasPreCondition) and modifies (hasEffect). Other properties can be added in order to express more requirements about business goals and constraints such as services costs, requirements, etc. All concepts involving both domain objects and state variables are also represented in the CIM’s domain ontology.

It is worth mentioning that other semantic specifications are proposed in the literature (e.g. [2]), but they concern with composition of concrete Web services rather than their characterization from a business viewpoint. Instead, our approach concerns with a very abstract view of services whose details are successively added in PIM and PSM levels.

As an example, the following axiom in the KB describes a generic Service that provides nearest geospatial objects:

\[
\text{NearestObject} \sqsubseteq \text{Service} \quad \exists \text{hasProfile.PosObject} \\
\text{PosObject} \sqsubseteq \text{Profile} \quad \exists \text{hasInput.GeoPosition} \quad \exists \text{hasOutput.GeoObject} \\
\text{PaymentLogic} \sqsubseteq \text{BLogic} \quad \exists \text{hasPrecondition.(User \\text{hasCredit} .>5000)}
\]

The basic building blocks in DL are concepts and properties, role assertions and individuals. The DL KB consists of TBox (Terminology Box) and ABox (Assertions Box) language features. The TBox contains concepts such as NearestObject and Service, and general properties of concepts such as hasInput. In addition, a few constructors help us to build complex concepts such as concept inclusion ($\sqsubseteq$), intersection ($\cap$), and existential quantification ($\exists$). We use the TBox feature both for describing services and queries and for reasoning in the CIM. On the other hand, concrete services are expressed as individual and role assertions in the ABox. For example, the service NearestCity1 is an individual of NearestObject (denoted in DL as NearestCity:NearestObject). Role assertions are mainly used to denote concrete service executions, for example: hasProfile(NearestCity1,Prof1), hasInput(Prof1, Pos123) and hasOutput(Prof1, NewYork), are possible role assertions for the service above.
The KM is completed with the abstract patterns. We have derived a set of abstract patterns suitable for web services [6], which provide several advantages with respect to common control patterns available in other languages. First, intentionality, we have maintained the number of redundant patterns to a minimum, in contrast to overlapping and alternative patterns present for example in WSBPEL, leading to simpler but complete set of abstract patterns. In addition, such abstract patterns are quite general to be independent enough of concrete control patterns used in current composition languages.

<table>
<thead>
<tr>
<th>Abstract Patterns</th>
<th>Axioms associated to abstract patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S = \text{SEQ}(A, B)$</td>
<td>$\text{hasOutput}^{-1}.A \land \text{hasInput}^{-1}.B$ is satisfiable $S \subseteq \text{Service}^* \land \exists \text{hasProfile}.(\exists \text{hasInput}.(\exists \text{hasInput}^{-1}.A) \land \exists \text{hasOutput}.(\exists \text{hasOutput}^{-1}.B))$</td>
</tr>
<tr>
<td>$S = \text{AND}(A, B)$</td>
<td>$S \subseteq \text{Service}^* \land \exists \text{hasProfile}.(\exists \text{hasInput}.(\exists \text{hasInput}^{-1}.A \lor \exists \text{hasInput}^{-1}.B) \land \exists \text{hasOutput}.(\exists \text{hasOutput}^{-1}.A \lor \exists \text{hasOutput}^{-1}.B))$</td>
</tr>
<tr>
<td>$S = \text{XOR}(A, B)$</td>
<td>The same as AND (Semantics included at PIM level)</td>
</tr>
<tr>
<td>$S = \text{AND-DISC}(A)$</td>
<td>The same as AND-DISC (Semantics included at PIM level)</td>
</tr>
<tr>
<td>$S = \text{OR-DISC}(c, A)$</td>
<td>The same as AND-DISC (Semantics included at PIM level)</td>
</tr>
<tr>
<td>$S = \text{IF}(c, A, B)$</td>
<td>The same as AND-DISC (Semantics included at PIM level)</td>
</tr>
</tbody>
</table>

We also provide abstract patterns for parallel processing (AND, XOR, OR) and simple loop-based patterns (COND-LOOP, ITER-LOOP). A special attention requires the AND-DISC and OR-DISC patterns. Both are based on the discriminator rationale given a list of services. Suppose that several services are available and perform the same action or task. Then, modelling these services with the discriminator pattern means that all services are executed in parallel but only the first service to complete its task is selected, while the remaining services are ignored. In our approach, we assume a relaxed version of the discriminator pattern in which candidate services are executed in sequence and not in parallel as does the original pattern. Then, the AND-DISC pattern means that if the first service fails, the pattern logic takes the second one in the list and so on, until any of the services is successfully completed. Anyway, this little change has only implications at PSM level, when a service composition description is transformed into an executable process, yet the discriminator pattern maintains its original meaning at higher levels (PIM and CIM). Interested readers can find in [6] a detailed description on how such set of abstract patterns is derived.

The abstract patterns described earlier can be semantically characterized to be used within the CIM’s KB for defining semantic service and queries. In contrast to other approaches, introducing abstract patterns at CIM level is a key aspect in our approach because it lets users describe more complex and structured queries. Users can not only
specify semantic queries in terms of IOPEs but also determine how candidate services should be combined. Table 1 shows the axioms associated to these abstract patterns.

5 Query Processing at CIM level

The CIM’s query language is aimed at obtaining semantic patterns that will be transformed into PIM objects. In the CIM, a query expresses the user goals, which are not only the IOPEs of the services but also the possible abstract pattern that they must satisfy. In this way, a query is composed of two parts: the specification of the services and the specification of the pattern according to the former. For example, the following query can be fetched to the CIM:

Query = 〈(SQ1 ⊑Service ⊓∃hasProfile.(∃hasInput.(X ∪Y) ⊓∃hasOutput.Z),
    SQ2 ⊑Service ⊓∃hasProfile.(∃hasOutput.W)),
    SEQ(AND-DISC(SQ1), SQ2) 〉

A query as above is resolved as follows:

1. By using the reasoner, we obtain all the concepts that satisfy the conditions expressed in the first part of the query.
2. If some concept cannot be satisfied (i.e. there not exists any service with the specified profile), then a semantic pattern is created for it taking into account its IOPEs. This semantic pattern follows the methodology in [10].
3. Merge the semantic patterns obtained in step 2 according to the specified abstract pattern.
4. Apply the semantics of the specified abstract pattern over both the satisfied concepts in step 1 and the sub-patterns of step 2.
5. Check that the intermediate states of all compositions are consistent with respect to the corresponding business logic conditions (hasPreCondition and hasEffect properties). This step will be further treated in future work since it requires rule-based reasoning mechanisms (e.g. SWRL, www.w3.org/Submission/SWRL/), which simulate the evolution of the state variables along the abstract pattern. This kind of reasoning is still an open issue within the Semantic Web research area.

Let us show an example of query resolution with the following KB:

S1 ⊑Service ⊓∃hasProfile.(Profile ⊓∃hasInput.X ⊓∃hasOutput.X')
S2 ⊑Service ⊓∃hasProfile.(Profile ⊓∃hasInput.Y ⊓∃hasOutput.Y')
S3 ⊑Service ⊓∃hasProfile.(Profile ⊓∃hasInput.X' ⊓∃hasOutput.Z)
S4 ⊑Service ⊓∃hasProfile.(Profile ⊓∃hasInput.Y' ⊓∃hasOutput.Z)
S5 ⊑Service ⊓∃hasProfile.(Profile ⊓∃hasInput.N ⊓∃hasInput.Z
 ⊓∃hasOutput.W)

After step 1, SQ1 is not satisfied and SQ2 is satisfied by service S5. As SQ1 is not satisfied directly by the reasoner, the Graph Handler builds a semantic pattern for SQ1 in order to discover possible execution plans that match the SQ1 (step 2). The resulting semantic pattern for SQ1 consists of {SEQ(S1,S3), SEQ(S2,S4)}. At this point it is important to mention the existing relation among the Graph Handler, the reasoner and the CIM’s KB. While the reasoner merely classifies the service
descriptions in the KB, the Graph Handler complements it by inferring automatically possible profiles using properly abstract patterns that fit the semantic query. Now in step 4 the reasoner checks that the profile inferred from the semantic pattern associated to SQ1 is compatible with the specification of SQ2. In this case, as the intersection of their input-output profiles is satisfiable (i.e. they intersect in concept Z), both services are satisfiable. Thus, the answer is the following semantic pattern:

\[
\text{SEQ(AND-DISC(\{SEQ(S1,S3),SEQ(S2,S4)\}),S5)}
\]

Notice that more than one answer is possible. For example, if more than one service satisfy SQ2 compatible with SQ1, then we obtain an answer for each one. Moreover, notice also that at this level we handle concepts not individuals (they are introduced at the PIM/PSM level). In fact, the answer above is the interface between CIM and PIM levels. As depicted in Figure 1, the resulting semantic pattern serves as input of the Matchmaker (within the PIM) which is the responsible to find semantic service descriptions that satisfy the requirements.

7 Conclusions and Future Work

The proposed ontology-driven architecture in a MDA-based framework allows us to take advantage of reusing CIM models for composing and reusing semantic web services. First, we have described a CIM’s DL KB capable of reasoning –classifying, composing, and querying– over services in an abstract way, and a querying-driven process at CIM level that achieves structured semantic answers given user goals, underlying the importance of the CIM models for improving the reuse of services and knowledge between CIM and PIM levels. Future work includes rule-based reasoning to simulate the evolution of state variables for checking that all compositions are consistent with respect to the corresponding business logic conditions.

References