Generating OWL-S Families by Utilizing Business Process Definitions and Feature Models

Umut Orhan and Ali H. Doğru

Middle East Technical University, Computer Eng. Dept., Ankara, 06531, Turkey

Abstract. This research introduces automated transition from domain models and process specifications to semantic web service descriptions in the form of service ontologies. Also, automated verification and correction of domain models are enabled. The proposed approach is based on Feature-Oriented Domain Analysis (FODA), Semantic Web technologies, and ebXML Business Process Specification Schema (ebBP). This approach is proposed to address the needs for achieving productivity gains, maintainability and better alignment of business requirements with technical capabilities in the engineering of service oriented applications and systems.

1 Introduction

Business Process Management (BPM) and Service-Oriented Architecture (SOA) combination is being promoted as a possible solution for achieving proper level of service abstractions and reaching the desired agility and responsiveness to changing business parameters. Unfortunately, in order to enable the anticipated BPM-SOA convergence, knowledge of the domain should be first transferred from domain experts to developers. This necessitates common means for shared vocabulary and understanding of the business requirements which can conveniently be achieved by utilizing ontologies. In this respect, we exploit the concept of service ontology as a bridge between business process models and service interface implementations.

It is desirable to generate service ontologies automatically, especially when we consider the complex and time-consuming nature of the ontology creation task. Thus, we chose to automate service ontology creation to some extent. In addition, we enable the Feature-Oriented Domain Analysis (FODA) [1] to manage differences and similarities across multiple service definitions: we produce not only a single service ontology but a set of related service ontologies.

An overall representation of this method is shown in Figure 1. We provide the domain experts with the necessary environment in order to define business process models and feature models. These defined domain engineering outcomes are then mapped to corresponding OWL-S instances. A resulting OWL-S instance may then be related to web service implementation in two ways; (1) provide requirements and eliciting constraints for the corresponding web service interface, (2) enable automatic discovery of semantically matching web services already implemented.

Major contributions of this research can be listed as follows;

DOI: 10.5220/0002201300420051

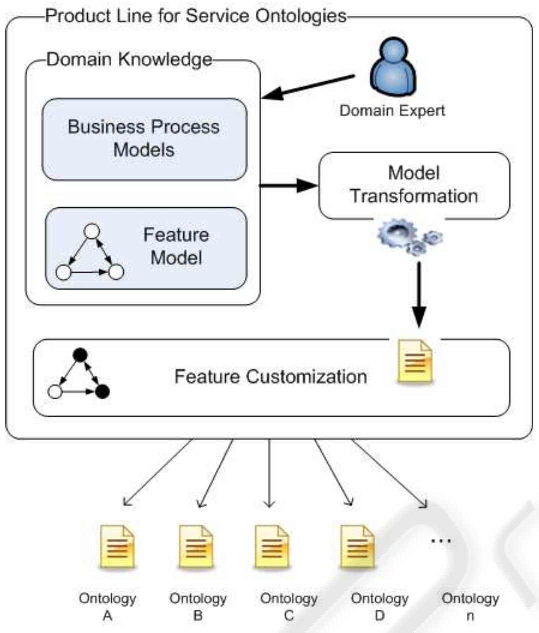

Fig. 1. Product line approach for generating a family of service ontologies.

- An OWL[1] ontology is developed to represent feature models. We also developed a set of SWRL[2] rules as axioms corresponding to relations among features. Verification and correction of feature model customizations are performed by enabling the JESS [2] which is a rule-based automated reasoner.
- We developed an ontology-aware feature model editor in order to facilitate visual development of feature models. The tool imports and exports feature models formally defined by our feature model ontology.
- We defined model transformation rules from ebBP[3] instances and feature models to accompanying service models. Generated service models are conceptualized as OWL-S[4] instances.

The introduced method is realized through the scope of an ongoing open source project called GENoDL[5] . The tool takes a feature model and an ebBP instance as input and refines them in order to generate OWL-S instances.

[1] OWL Web Ontology Language, http://www.w3.org/TR/owl-ref/
[2] SWRL: A Semantic Web Rule Language Combining OWL and RuleML, http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/
[3] ebXML Business Process Specification Schema, http://docs.oasis-open.org/ebxml-bp/2.0.4/OS/
[4] Semantic Markup for Web Services, http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/
[5] Automated Service Ontology Generator Tool based on Description Logics, http://sourceforge.net/projects/genodl/

In order to define ebBP instances, the ebBP Editor [3] tool is integrated. It was previously developed within the scope of a research project funded by the European Commission. The domain expert can model business processes in Business Process Modeling Notation (BPMN) [4] and the tool then exports these models to accompanying ebBP definitions.

The remainder of this paper is organized as follows. Section 2 summarizes the related work on service ontology generation methods. In Section 3, we present our semantically enriched feature modeling approach enabled in generating OWL-S instances. The transformation method from ebBP to OWL-S is given in Section 4. An example walkthrough with the implemented tool is described in Section 5. Finally, Section 6 concludes the paper and presents the future work.

2 Related Work

Various work has been conducted [5, 6, 7, 8, 9] that partially cover our ambitions. Usually, automation has been missing and also our mass customization of a set of related ontologies was not addressed.

In this work, we separate the identification, specification and realization concerns of a web service. Business processes for identification, service ontologies for specification, and service implementations for realization. This separation of concerns, which well fits in Service-Oriented Modeling and Architecture (SOMA) [10], allows us to automate the mapping from identification step to specification step through semantic web technologies. This also provides means for capturing commonalities and variabilities in service models by exploiting feature models.

3 Semantically Enriched Feature Models

Basically, a feature model provides a hierarchical organization among features within a concept. In addition to *parent-child* relation, a simple feature model can define *mandatory*, *optional*, *alternative*, *excludes*, *requires* and *OR* relations among features. In our study, OWL is decided to be used for feature model formalism because it is a well known standard in the area and is supported by various technologies, tools and development environments like Protege-OWL[6], SWRL and JESS. In this respect, OWL reasoning engines such as JESS can be deployed to check for inconsistencies within a feature model and correct them automatically.

Firstly, we define a *Feature* class having two object properties, *hasParentFeature* and *hasChildFeature* respectively, which are transitive properties that are inverse of each other. These properties are required to express IS-A relations in structured view or OR relations in common feature model understanding. The concept of the feature model can be considered as an ordinary feature which has no parent feature.

In order to fully represent the mandatory and alternative relations, we derive two specific child classes from the *Feature* class. Each class is a subclass of the *owl:Thing*

[6] Protege-OWL, http://protege.stanford.edu/overview/protege-owl.html

class. We assert that *Alternative Feature* and *Mandatory Feature* as mutually disjoint. An overview of the classes defined within the Feature Model Ontology is given below.

$$Feature \sqsubseteq \top \tag{1}$$

$$AlternativeFeature \sqsubseteq Feature \tag{2}$$

$$MandatoryFeature \sqsubseteq Feature \tag{3}$$

$$AlternativeFeature \sqcap MandatoryFeature = \bot \tag{4}$$

A number of axioms for checking the consistency of the feature model customizations are formally defined within the scope of this work. In general, these axioms are exploited not only for verifying the feature model but also correcting any inconsistency found. These axioms are described as SWRL rules. They are normally stored as OWL individuals which can refer to the resources within the associated knowledge base. Class definitions of these OWL individuals are introduced in SWRL ontology. Defined axioms and their SWRL implementations are given below;

1. A feature cannot be selected unless its parent feature has been selected already. For all x and y;

$$Feature(?x) \wedge isSelected(?x, false) \wedge hasChildFeature(?x, ?y) \wedge \\ isSelected(?y, true) \rightarrow isSelected(?y, false)$$

2. A mandatory feature must be selected whenever its parent feature has been selected. For all x and y;

$$Mandatory\_Feature(?x) \wedge hasParentFeature(?x, ?y) \wedge isSelected(?y, true) \\ \wedge isSelected(?x, false) \rightarrow isSelected(?x, true)$$

3. Only one feature must be selected among its alternatives. For all x and y;

$$Alternative\_Feature(?x) \wedge isSelected(?x, true) \wedge alternativeOf(?x, ?y) \\ \wedge isSelected(?y, true) \rightarrow isSelected(?y, false)$$

4. A feature is selected whenever the other feature which requires it has been selected. For all x and y;

$$Feature(?x) \wedge requires(?x, ?y) \wedge isSelected(?x, true) \rightarrow isSelected(?y, true)$$

5. A feature is deselected whenever the other feature which excludes it has been selected. For all x and y;

$$Feature(?x) \wedge excludes(?x, ?y) \wedge isSelected(?x, true) \rightarrow isSelected(?y, false)$$

The customized feature model has no variability points but a definite set of selected features. In order to verify a customized feature model represented with OWL instances, we employ our feature model axioms, Protege-SWRL plugin, and the JESS rule engine. Protégé-SWRL plugin has no reasoning capability, however, it supports

API level integration with existing rule engines such as JESS. The plugin can translate SWRL rules to the JESS rule language. The necessary transformation methods from OWL and SWRL concepts to JESS constructs and vice versa are given in [11]. Once the OWL and SWRL concepts are transformed to JESS context, the execution engine can perform reasoning. The overall view of the integration between SWRL, Protégé-OWL and JESS rule engine is given in Figure 2. The implementation of the GENoDL and Protégé-SWRL integration is also given in [12].

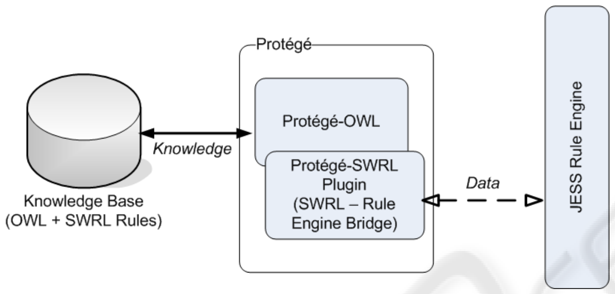
Protégé
Protégé-OWL
Knowledge
Protégé-SWRL Plugin (SWRL – Rule Engine Bridge)
Data
JESS Rule Engine
Knowledge Base (OWL + SWRL Rules)

Fig. 2. Reasoning engine integration through the Protégé-SWRL adapter.

3.1 Mapping Customized Features to OWL-S

OWL-S is an emerging de-facto semantic web standard that supports automation of various web service related activities such as service discovery, composition, execution and monitoring. OWL-S provides process models of atomic as well as composite web services.

During the service ontology generation, the selected (or in other words customized) nodes of the feature model are automatically transformed into accompanying OWL-S constructs which is in this case the *serviceParameter* element of OWL-S Profile. OWL-S provides an unbounded list of service parameters that can contain any type of information. Thus, the *serviceParameter* construct is very suitable for representing feature customizations. A *serviceParameter* consists of two attributes *serviceParameterName*, the name of the actual parameter, which could be just a literal, or perhaps the URI of the process parameter, and *sParameter* which points to the value of the parameter within some OWL ontology. A customized feature placed as a leaf node in the feature model can be mapped to a *serviceParameter* instance with the *Condition* type for the *sParameter*.

Moreover, new features can be included under the related variability category by the domain experts in order to extend the scope of the default web service variability model. After a new feature is added to the model, it can be transformed into the service ontology definition as it is explained for the default features. Variabilities captured within a feature model can be reflected to a business collaboration's OWL-S counterpart which is *Service*.

3.2 A Sample Feature Model for Web Services

We analyze the variability in web service definitions from three broad perspectives namely *Service Grounding*, *Service Profile*, and *Service Model* which have been once introduced by the OWL-S service ontology. We adopt the Jiang et al's notion [13] of families of web services and bring the feature-oriented domain analysis (FODA) to it instead of the pattern-based variability management approach.

In general, Service Grounding describes how to access the service through concrete specifications such as binding protocol, address, message formats etc. Main variability points of service grounding are identified in [14] as *Binding Protocol* and *Binding Time*.

Service Model describes the semantics of how a service interacts with its clients, and the data and control flows of corresponding process specification. OWL-S process models; Simple, Atomic, and Composite are subclasses of the Service Model. The ways a client may interact with a service through exchanging messages provides a basis for Service Model variability.

Service Profile describes what is done by the service and presents necessary information such as service name, its text description, and contact information. Service profiles are generally enabled in automated operations like dynamic service discovery. It can be considered as a yellow page entry for the service functionality. Information about inputs, outputs, preconditions and effects of the service are given the profile part. One important aspect of the service profile is its service parameter option which give the characteristic features of the service such as QoS and classification of service functionality in taxonomies provided by service registries. OWL-S's service profile can be directly mapped to UDDI registry data model [15].

In order to produce appropriate exceptions, the ebBP specifications mandate a business service interface to conform to a number of service parameters such as *AuthorizationRequired*, *NonRepudiationRequired*, and *NonRepudiationOfReceiptRequired* during the execution of the corresponding business activity. Indeed, each of these parameters create a source of variability in service profiles.

We expose the previously identified variability points to our semantically enriched feature model as shown in Figure 3.

4 Mapping ebBP to OWL-S

In this study, ebXML Business Specification Schema (ebBP) is exploited in capturing process definitions and business choreographies from a high abstraction level where contributions of domain experts can be effectively incorporated in devising the domain model. The main advantage of using ebBP is its powerful built-in mechanisms for separating the definition of the process model from its realization and making it independent from its enablers namely business actors.

ebBP is capable of specifying process model parameters for configuring service interfaces to execute and monitor business collaborations. However, ebBP does not specify how to associate a defined service interface to its real world implementation.

In this research, conceptual mapping schemes between the generic ebBP instances and OWL-S ontologies are developed. As an outcome of this mapping, previously defined business processes are refined and brought one step closer to the realization phase

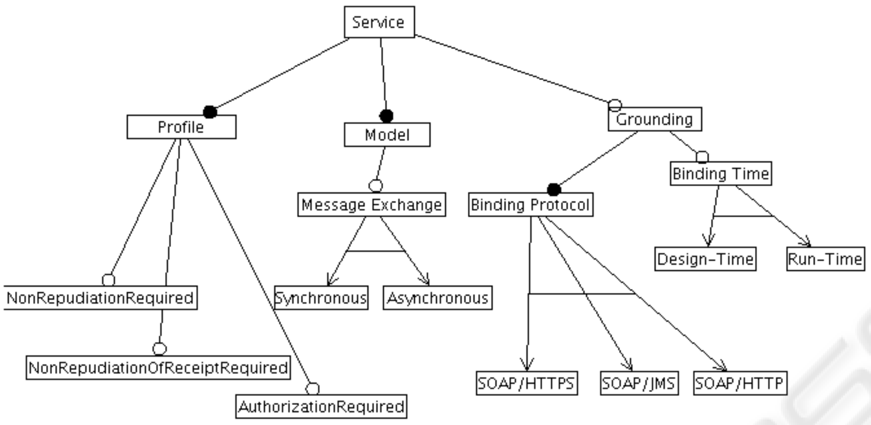

Fig. 3. A reference variability model for semantic modeling of web service families.

automatically. Moreover, mapping ebBP definitions to OWL-S may foster service reuse. Once the OWL-S model of a business service interface is described then the service implementation may be discovered from the existing assets instead of developing the service every time from scratch.

The basis for our mapping method is formed through comparing common components sharing similar semantics in both standards. Also, we adopt ebBP's bottom up design approach for describing business collaboration and propose our transformation strategy as follows;

1. Transform Business Transactions
2. Transform Business Document Flow for Business Transactions
3. Transform Binary (Business) Collaboration re-using the mapped Business Transactions
4. Transform the choreography for the Binary (Business) Collaborations
5. Transform higher level Business Collaborations re-using the lower level Business Collaborations translated previously

An overview of the mapping specification is depicted in Figure 4. The conceptual mappings from ebBP elements to their OWL-S counterparts are listed in [12]. However, due to the size limitation we can only provide an essence of the transformation method which is for the mapping from ebBP's *BusinessCollaboration* to OWL-S's *CompositeProcess*. According to the ebBP specifications, a choreography is an ordering of business activities within a business collaboration in order to specify which business state is expected to follow another state.

Basically, OWL-S's composite process consists of other atomic or composite processes. Control flow of a composite process is specified using control constructs which can be nested to an arbitrary depth. Like business collaborations in the ebBP, composite processes can be considered as state-oriented workflows. In the transformation method, *CompositeProcess* class is preferred for representing underlying semantics of business choreography. Main control mechanisms of the whole choreography is the *Sequence*

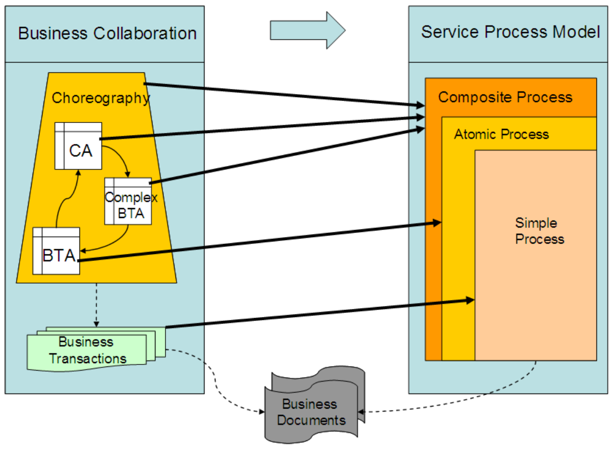

Fig. 4. Overview of the mapping specification.

element. Linking constructs (*FromLink* and *ToLink*) are mapped according to their type and the class of the state they refer. For example, *FromLink* is transformed into a *ControlConstruct* that can be further specified as a *Perform*, *Split*, *Split-Join* or *Choice* based on the type of the referred state; *Business Transaction Activity*, *Fork*, *Join* and *Decision* respectively. In ebBP, a choreography starts by linking to a business state so, we can associate *Start* with a *Sequence* instance whose *list:rest* element refers to the state that *ToLink* of the *Start* linking as well. Overall mapping from choreography construct of ebBP to their OWL-S counterparts are given in Table 1. *Transition*, *Fork*, *Join* and *Decision* have at least one *FromLink* and one *ToLink* but maximum occurrence of these linking constructs can vary depending on the choreography type. *Fork* and *Decision* include at least two *ToLink* on the other hand, *Join* has at least two *FromLinks*.

Table 1. Choreography to Service Process Model.

ebBP Choreography	OWL-S Counterpart (ControlConstruct)
/FromLink	/ControlConstructList/list:first/ControlConstruct
/ToLink	/ControlConstructList/list:rest/ControlConstructList
/Start	CompositeProcess/@composedOf/Sequence
/Start/@nameID	/Sequence/@rdf:ID
/Transition/@nameID	ControlConstruct/@rdf:ID
/Fork/@nameID	/Split/@rdf:ID
/Join/@nameID	/Split-Join/@rdf:ID
/Decision/@nameID	/Choice/@rdf:ID
/B.T.A	/C.C.L./list:first/Perform/AtomicProcess
/Success	No suitable match
/Failure	No suitable match

5 An Example Walkthrough with the GENoDL

The first step in generating web service ontologies is to provide our reasoning tool with an ebBP instance as input. The tool then refines business process constructs such as *BusinessCollaboration* in order to compile them in possible service ontology representations. To start the transformation process, first a *BusinessCollaboration* is selected from the *Business Process Manager* section of the user interface. The tool then loads the default service feature model to the feature model editor. If the domain expert has any extension or further customization requests, she can modify the feature model by inserting/deleting or (de)selecting features through using the feature model editor bundled within the tool. Newly inserted features should be associated with their formal and machine readable definitions. After modifications, reasoner checks the resulting feature model's consistency and automatically corrects any invalid customization with domain expert's empowerment. Finally, the verified feature model along with the selected business collaboration specification is transformed into OWL-S notation according to the given transformation rules.

It can be easily understood that a family of related service ontologies can be quickly generated by applying the proposed generative method. For example, by selecting different children of the *BindingProtocol* feature can result in various service conceptualizations that each of their implementations will be subject to be used in a distinct application scenario.

6 Conclusions and Future Work

The developed mechanism was experimented through an example as in [12]. This example included the mapping from an existing business process model, that represents the domain-level knowledge. The outcome has been promising: in general it has been observed that automated generation of web service ontologies is possible and usable. However, our observation also states that very high-abstraction level elements of the domain are not easy to map directly.

Our novel generative method for service ontology creation has not reached its maturity yet. We would also like to address coarser-grained service ontologies and more detailed process models. Also, the generative method will be extended with the mapping rules of other ebBP concepts such as guards, exceptions and signals.

Current versions of the feature model editor and feature model ontology do not support cardinality-based relations among features as proposed in [16]. This is another possible improvement area. An evaluation of memory allocated for varying input sizes will be performed in order to assess the feasibility of the feature model verification operation from a different perspective.

Although our product line approach for service ontology generation is based on two well known standards (ebBP and OWL-S), a more generic approach which is independent from implementation technologies is considered as future work. Finally, evaluation of the quality and domain coverage of the resulting service ontologies should be explicitly justified through anticipated metrics and methods, so that domain experts and developers can assess them easily.

References

1. Kang K. C., et al.: Feature-oriented domain analysis (foda) feasibility study. Carnegie-Mellon University Software Engineering Institute (1990)
2. Jess: the rule engine for the java platform http://www.jessrules.com/
3. Ride deliverable 5.3.1 - contribution to standards: ebbp editor v1.0.4 user manual http://-www.srdc.metu.edu.tr/webpage/publications/2007/ebBP Editor UserManualv1.0.4.pdf
4. Business process modeling notation (bpmn), Object Management Group/Business Process Management Initiative http://www.omg.org/docs/dtc/060201.pdf
5. Czarnecki K., Kim C. H. P., Kalleberg K. T.: Feature models are views on ontologies. IEEE SPLC (2006)
6. Lee S., et al.: An approach to analyzing commonality and variability of features using ontology in a software product line engineering. IEEE SERA (2007)
7. Wang H., et al.: A semantic web approach to feature modeling and verification. SWESE (2005)
8. Guo L., et al.: Mapping a business process model to a semantic web service model. ICWS ’04: Proceedings of the IEEE International Conference on Web Services (2004)
9. Shen J., et al.: From bpel4ws to owl-s: integrating e-business process descriptions. SCC ’05: Proceedings of the 2005 IEEE International Conference on Services Computing (2005)
10. Arsanjani A.: Service-oriented modeling and architecture (SOMA). IBM developerWorks (2004)
11. O’Connor M., et al.: Supporting rule system interoperability on the semantic web with swrl. Fourth International Semantic Web Conference (2005)
12. Orhan U.: Automated service ontology generation from domain engineering outcomes http://genodl.wiki.sourceforge.net/space/showimage/genodl.pdf
13. Jiang J., Ruokonen A. and Systa T.: Pattern-based variability management in web service development. ECOWS ’05: Proceedings of the Third European Conference on Web Services (2005)
14. Segura S., et al.: A taxonomy of variability in web service flows. Service Oriented Architectures and Product Lines (SOAPL - 07)
15. Luo Jim, et al.: Adding owl-s support to the existing uddi infrastructure. International Conference on Web Services ICWS’06, pp. 153–162 (2006)
16. Czarnecki K., Helsen S., Eisenecker U.: Staged configuration through specialization and multi-level configuration of feature models. Software Process Improvement and Practice, special issue on ”Software Variability: Process and Management, vol. 10(2), pp. 143 – 169 (2005)