A Method for Automatic Generation of OWL-S Service Ontology

Jin-Hyuk Yang*, and In-Jeong Chung*

Abstract: We present in this paper the methodology for automatic generation of OWL-S service model ontology along with the results and issues. First, we extract information related to atomic services and their properties such as IOPE from the UML class diagram, and retrieve information related to the composition of services from the UML state-chart diagram. Then, the XSLT applications utilize the acquired information to generate the OWL-S service model ontology through the predefined mappings between OWL-S constructs for composite services and UML state-chart primitives. For the justification of generated service ontology, several validation checks are performed. Our service ontology generation method is general and fully automatic, as well as effective, in that it is achieved in an environment familiar to developers, and information needed to generate service ontology is provided necessarily during service development. It is also noticeable to facilitate representing the condition with GUI rather than a complex language such as OCL.

Keywords: Ontology, Semantic Web, OWL-S, State-chart, and UML

1. Introduction

As the role of the Internet becomes greatly valued, not only for static resources such as web pages, but also for dynamic ones such as services, the importance of web services has been emphasized. However, the use of the fundamental technologies of web services, such as WSDL (Web Service Description Language)[1], UDDI (Universal Description, Discovery, and Integration)[2], and SOAP (Simple Object Access Protocol)[3], lacks intelligent utilization of services called semantic web services[4]. For example, as WSDL documents describe the interface of service only, information required to composite and interoperate among services is not contained in a WSDL document. As UDDI registries support keyword-based APIs for searching services, flexible results cannot be provided. Hereupon, various standards have been published by different communities such as BPEL4WS (Business Process Execution Language for Web Services)[5] in the web service community, WSCI (Web Service Choreography Interface)[6] by W3C, and OWL-S (OWL for Services)[7] in the semantic web community. Among these specifications, OWL-S was adopted as the semantic web service markup in this paper, since OWL-S, as service ontology written in OWL (Ontology Web Language)[8], enables inferencing and allows the use of services more intelligently. Relationships with the other various specifications can be found in [9] and [10].

In this paper, we present the methodology for automatic generation of OWL-S service ontology, along with the results. In particular, we focused on the OWL-S service model ontology among three OWL-S ontologies (service profile, service model and service grounding), since crucial information on how to interoperate with other services is described within the service model ontology. This kind of information is essentially required to enable intelligent web services such as automatic web services composition.

For automatic generation of the OWL-S service model ontology, we extracted information related to atomic services and their properties such as IOPE (Input, Output, Precondition, and Effects) from the UML class diagram, and retrieved information related to the composition of services from the UML state-chart diagram. Then, XSLT (Extensible Stylesheet Language Transformation) applications utilized the acquired information to generate the OWL-S service model ontology through the defined mappings in section 3 between OWL-S constructs for composite services and UML state-chart primitives. For the justification of generated ontology, we performed a few available validation checks. The rationales behind our approach are described in detail in section 3, and the difference with existing UML approaches is addressed in subsection 2.2.

Our service ontology generation method is not only fully automatic but also general, to be applied in any case, and is also effective in that it is performed in an environment familiar to developers, and information needed to generate service ontology is necessarily provided during service development. This familiar environment means they use only UML and do not need to use OWL-S to generate OWL-S ontology. In addition, we propose a method for modeling OWL-S condition expression with GUI in the UML diagram instead of a complex language such as OCL (Object Constraint Language). A detailed explanation on this issue is addressed again in subsection 3.2.

Manuscript received April 4, 2006; accepted May 22, 2006.
Corresponding Author: In-Jeong Chung
* Dept. of Computer Science, Korea University, 208 Seochang-Li, Chochiwon-Eup, YunkiGun, ChoongNam, Korea (grjinh@korea.ac.kr, chung@korea.ac.kr)

The organization of this paper is as follows: Section 2 describes the semantic web services and introduction to OWL-S, in brief. Also, the existing works for ontology generation are addressed. Automatic generation of the OWL-S service model ontology method is presented in section 3. Implementation is presented in section 4, and analysis and discussion are mentioned in section 5. Finally, the conclusion and further works are described in section 6.

2. Related Works

2.1 Short Introduction to Semantic Web Services and OWL-S

Semantic web services, often called 'intelligent web services', first introduced in [4], enables web services to be intelligent using ontologies which play an important role as metadata for inferencing in semantic web. One reason for semantic web services gaining recognition can be attributed to limitation on intelligent utilization of web service via its fundamental technologies such as WSDL and UDDI. WSDL only exposes the information about service's signature and does not describe the specific information related to service's behavior. And, UDDI makes it difficult for various users with different needs to retrieve flexible and appropriate services, as it supports keyword-based search APIs.

As a part of the efforts to overcome WSDL's expressiveness problem, expressive markup languages (mostly for composition) such as BPEL4WS, WSCI and OWL-S have been published. Among these, OWL-S is service ontology written in OWL. OWL is an AI-inspired markup language and supports reasoning. Also, it has been adopted by W3C as the standard for representing ontology. Relationships and comparisons between OWL-S and other various specifications can be found in [9] and [10].

The goal of OWL-S is to allow automatic services discovery, execution, composition and interoperation. As metadata for a service, OWL-S has three sub-ontologies for describing the service: Service Profile ontology, Service Model ontology and Service Grounding ontology. Service Profile describes what the service does, including IOPE information, and is used for automatic services discovery. Service Model describes the conditions and constraints on the service's behavior, and Service Grounding describes how to access the service.

2.2 Existing Ontology Generation-related Works

The main idea behind using ontology is to pursue automation and intelligence via reasoning on metadata with regard to resources. However, the task of creating ontology is time-consuming and difficult as indicated in [11]. Therefore, automatic and effective ontology creation is very important.

[12] describes an ontology creation method for database design with some heuristics. [13] applies machine learning and statistical techniques to automatically extract ontology knowledge from documents. [14] uses hierarchical clustering to generate the concept hierarchy, and then generates RDFS ontology. [15] uses a decision tree and a set of rules acquired from the ID3 algorithm. [16] uses a knowledge grid applied to the existing grid with data mining techniques.

On the other hand, there are UML approaches for ontology modeling and generation. [17] addresses the use of a UML class diagram for ontology modeling. [18] mentions ontology modeling for agent through comparisons with description logic, which is the origin of ontology markup. In particular, [19], as white paper adopted in OMG ontology working group, proposes the issues and solutions that occurred in modeling ontology with UML. Also, [20] indicates these kinds of problems. In [19] and [20], mapping definitions are described between UML primitives and ontology markup languages such as OIL and DAML+OIL. The recent MDA (Model Driven Architecture)-based method for ontology modeling and generation is introduced in [21] and [22]. [23] mentions the current situation and future issues of UML approaches, including reasons for using UML.

The main difference between our approach and the existing ones is to not create domain ontology, but generate service ontology. As described above, the domain ontology creation approach can be divided into two kinds[1]: The approach in the first category extracts the concepts and relations from a number of documents (mostly unstructured) or database using domain analysis or data mining techniques. The UML approach in the second category uses a UML class diagram to model the domain concepts and relationships, and then transforms to concrete ontology markup automatically. However, modeling domain ontology with a UML class diagram involves manual processes (who will extract the domain concepts and relationships?). On the other hand, the process of service ontology generation focuses on the extraction of the service's interface and service's composition infor-mation required to be interoperable, rather than domain concepts and relationships from a number of unstructured documents. Note that our approach differs from the existing UML modeling paradigm, though UML diagrams are used in modeling; the UML class diagram and UML state-chart diagram are necessarily created during the service development lifecycle (developers first design the model with UML and then apply coding), regardless of generating the service ontology. Namely, extra effort and time to model service ontology is not needed.

It can be found in [24] as related work on creating OWL-S service ontology. [24] uses service's auto-

[1] Note that two kinds of methods are not exclusive, as UML modeling approach simply mentions a representation scheme and has nothing to do with any method of extraction itself. The extraction approach may rely on the UML approach as its representation for the extracted concepts and relations. Once done with UML modeling, various techniques such as a transformation method can be applied to create ontology based on the model.

generated WSDL document and annotates it. [25] uses a UML activity diagram to generate service's BPEL4WS specification. However, [24] is semi-automatic, and as such, is an annoyance to the developers. [25] is similar to our approach with UML activity diagram and BPEL4WS, rather than UML state-chart diagram and OWL-S. Comparisons between OWL-S and BPEL4WS are described in [9] and [10], and the reasons for using a UML state-chart diagram instead of an activity diagram is described in section 5.

The most remarkable approach is [42], which automatically generates OWL-S ontology from the WSDL document through the mappings between OWL-S and WSDL. However, [42] enforces manual processing for completing the OWL-S service model ontology when it contains more than one atomic process (i.e., when the service is a composite process). In fact, this is due to the expressiveness of WSDL. Therefore, our research can be regarded as a complementary approach to [42]. Moreover, we present a way to express the condition with GUI instead of a complex language such as OCL.

3. Automatic Generation of OWL-S Service Ontology

3.1 Motivation and Design Principles

Service ontologies must be created in order to realize semantic web services. If it is expensive and time-consuming to create the service ontologies, this becomes an obstacle to populating ontologies. Therefore, it is desirable to create service ontologies that are automatic and effective. Even some tools such as [26] and [27] provide an ontology-editing environment, and they force modelers (developers) to understand OWL-S. Moreover, extra time and efforts are needed in modeling ontologies. Hereupon, we decided on two design principles.

***Principle 1**: Service ontology must be generated in an automatic manner.* To populate the service ontologies necessarily needed in the realization of semantic web services, it is highly desirable to generate the service ontologies in automatic form, as services' WSDL documents are automatically generated in Java and .NET environment. In other words, information related to existing services (or ones being created) should be contributed to generate the service ontologies without additional use of tools or extra time and costs.

***Principle 2**: Generation of service ontologies should be easy and familiar to creators of service ontologies.* Most developers (service ontology creators) are unfamiliar with OWL-S. It is a huge obstacle to populating service ontologies, to have them create ontologies with an unfamiliar markup language, not to mention syntax errors; the task of generating the service ontologies is additional work for them (they are usually programmers not AI experts and need only to create services themselves). If the task is difficult, they will be reluctant to create the service ontologies. Therefore, the task must be performed in an easy and familiar environment for developers.

We embody the following two approaches to reflect the above principles.

We considered the XSLT application to generate the service ontology. XSLT application takes an XML file as input and can generate any type of document. It also allows selecting, sequencing, arranging, correcting, adding, sorting, and filtering XML data precisely at one's convenience. In addition, it is possible to access most XML document components (element, attribute, comment, and processing instruction). Moreover, it is allowed to use variables and built-in functions as in a procedural programming language. The main factor behind our decision to use XSLT to generate service ontology automatically is as follows: Acquired information related to service model at the UML approach (as our second decision principle, described below) is exported to XMI (XML Metadata Interchange), and this is the XML document; XMI is an OMG standard for exchanging metadata.

In order to provide an easy and familiar generation environment for developers, we chose UML, which is widely adopted in software engineering as the GUI standard for modeling. UML and UML CASE (Computer Aided Software Engineering) tools are familiar to service developers, as they design and analyze via UML and UML CASEs. It is our main objective to extract information needed to generate service ontology without any additional effort or time from UML diagrams which are necessarily created during the service development process. In order to accomplish this objective, UML diagrams are exported to XMI files which are used to generate service ontology in XSLT applications through the defined mappings below.

3.2 Mapping Definitions Between OWL-S Constructs for Composite Services and UML State-chart Primitives

In this section, we will address several rules embodied in the XSLT application used to generate the OWL-S service model ontology. These rules are based on the mappings between OWL-S constructs for composite services and UML state-chart primitives.

We considered two separate processes in generating the OWL-S service model ontology: The first process of generating atomic services and their IOPE-related attributes, and the second process of generating information related to composite services. In other words, information related to services' attributes is extracted from the UML class diagram in the first process, and composition information related to composite service is extracted from the UML state-chart diagram in the second process. The reason why we extracted different information from different UML diagrams is because the UML class diagram

is suitable for describing the atomic services, as well as their attributes and relationships with other atomic services, while the UML state-chart diagram is suitable for model services' behavior and allows a description of composite service composed of other composite services. Namely, the UML class diagram cannot provide what the UML state-chart can, and vice versa.

In the following, we only define the mappings between OWL-S constructs for composite service and UML state-chart diagram primitives, since mappings between primitives of the UML class diagram and OWL-S's attribute-related information are simple and already defined in [17-19].

- *Sequence*: OWL-S Sequence is a construct for specifying the sequence of services. It is defined as stereotyped[2] *transition*.

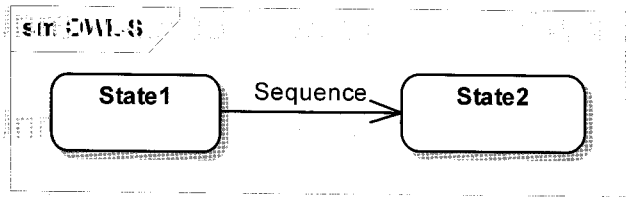


Fig. 1. Mapping definition for OWL-S Sequence

- *Split and Split+Join*: OWL-S Split and Split+Join are constructs for modeling synchronization. They are defined using the Fork/Join primitive.

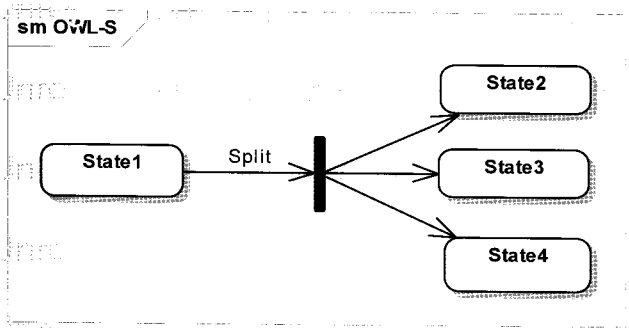


Fig. 2. Mapping definition for OWL-S Split

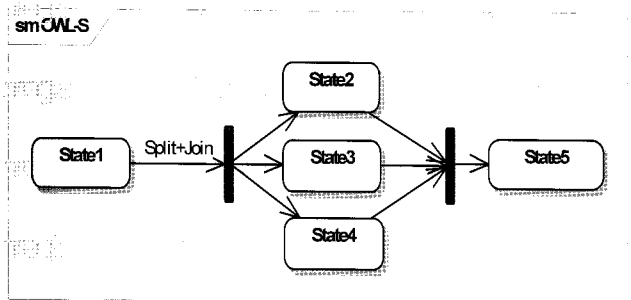


Fig. 3. Mapping definition for OWL-S Split+Join

- Choice and AnyOrder: OWL-S AnyOrder and Choice are constructs for modeling for selections. They are defined using Choice primitive.

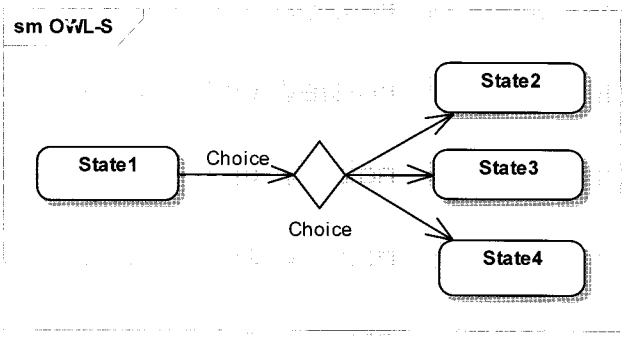


Fig. 4. Mapping definition for OWL-S Choice

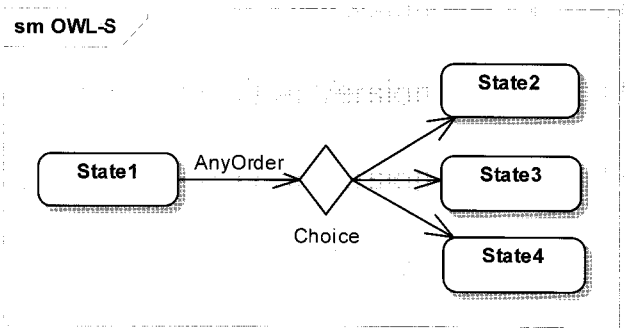


Fig. 5. Mapping definition for OWL-S AnyOrder

- *If-Then-Else*: OWL-S If-Then-Else is a construct for modeling conditional branching. It is defined using Choice for branching. In addition, stereotyped *class* and *dependency* are used.

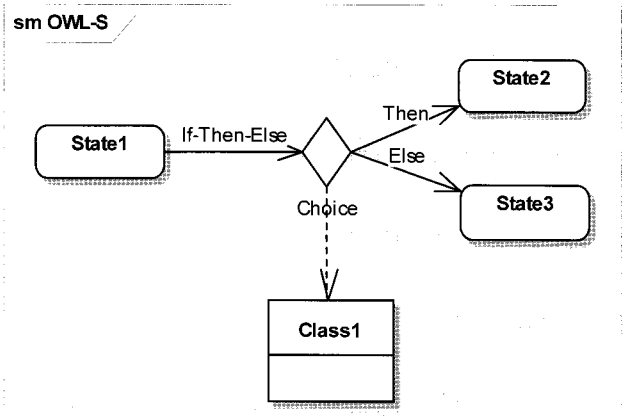


Fig. 6. Mapping Definition for OWL-S If-Then-Else

- *Iterate, Repeat-While and Repeat-Until*: OWL-S Iterate[3] and its subclasses Repeat-While and Repeat-Until are structured loop constructs. Repeat-While and Repeat-Until are defined as combinations of mapping definition used for OWL-S If-Then-Else (for specifying condition) and stereotyped *transition* primitive. Repeat-Until is defined in the same manner.

[2] UML profile for OWL-S constructs is described in subsection 5.3.

[3] This construct is abstract class, so we did not consider it when defining mappings.

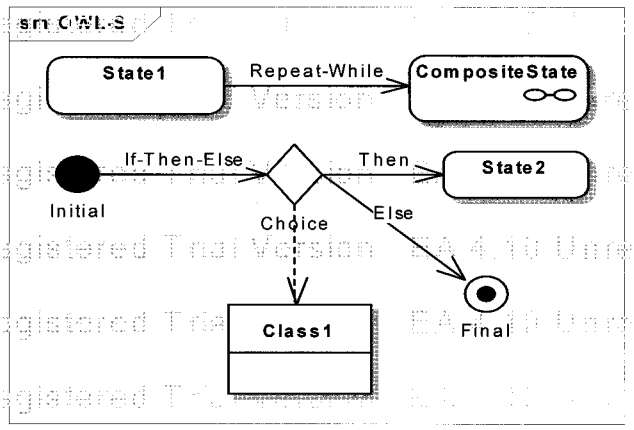


Fig. 7. Mapping definition for OWL-S Repeat-While

It is noticeable in the mapping definitions how the OWL-S If-Then-Else construct is mapped to UML state-chart diagram primitives. The reason for using *class* and *dependency* beyond UML state-chart diagram primitives is because of our decision to use GUI to represent condition. Allowed languages for expressing condition in OWL-S include SWRL (Semantic Web Rule Language)[28], RDF (Resource Description Framework)[29], KIF (Knowledge Interchange Format)[30], and PDDL (Planning Domain Definition Language)[31]. Among these specifications, we chose SWRL, since it is not only layered on top of OWL, but is also considered the candidate standard for rule expression by DAML.org. Atoms of SWRL can be created using unary predicates (classes), binary predicates (properties), equalities and inequalities. These SWRL atoms are children of ruleml:_body and ruleml:_body which have ruleml:_imp as their parent component in RuleML [32]. Among the many constructs for various SWRL atoms, we chose three first of all: classAtom, individualPropertyAtom, and builtinAtom. However, others can be similarly modeled in our approach.

Again, here is another reason for using *class* and *dependency* to model OWL-S condition expressed with SWRL. Condition (predicate) name and information related to arguments and their types are needed to specify the condition using the three above SWRL atoms. Some of our considerations on describing the necessary information for specifying condition in the UML state-chart diagram include use of OCL and direct representation of SWRL expression within note section. However, using OCL or specifying note is forcing developers to understand OCL and SWRL, and this approach is not considered automatic because the represented condition expression in OCL or SWRL must be parsed and manipulated again. In order to solve this problem, we decided to use *class* and *dependency*. Stereotyped *dependency* was used to represent the SWRL atom type, while the stereotyped *class* was used to describe the SWRL atom name. Attributes are used to describe the condition's arguments and their types within the class. This approach makes it easy to express condition with GUI in a UML state-chart diagram, and allows transformation to be automatic and simple.

3.3 Transformation Algorithm

Based on the mappings in the previous subsection, we address in this subsection transformation algorithm of XSLT application, which takes the XMI file exported from the UML state-chart diagram and generates one of the two parts of the OWL-S service model ontology. The algorithm on XSLT application for generating the remaining two parts of the OWL-S service model ontology is not described here because it is simple and published methods can be applied in this case, as we mentioned in subsection 3.2.

Step 1: **Extract** and **output** the entire composite service name.
Step 2: Perform the following **for each** case.
  Step 2-1: **Case**: Sequence
    // Find and print source and target state of Sequence in order.
    Identify source id and target id of Sequence transition in XMI tree structure, and then **output** corresponding names of states, respectively.
  Step 2-2: **Case:** Split and Split+Join
    // Find and print source state and associated target sates of Split and Split+Join.
    Identify state (Fork/Join primitive in UML state-chart diagram) having target of Split (Split+Join) transition as source in XMI tree structure, and then **output** the names of target states of transitions having id of identified state as source.
  Step 2-3: **Case:** AnyOrder and Choice
    // First, find and print source of AnyOrder (Choice)
    // Second, find and print associated targets of Choice (UML)
    Identify state (Choice primitive in UML state-chart diagram) having target of AnyOrder (Choice) transition as source in XMI tree structure, and then **output** the names of target states of transitions having id of identified state as source.
  Step 2-4: **Case:** If-Then-Else
    Identify state (Choice primitive in UML sate-chart diagram) having target of If-Then-Else transition as source in XMI tree structure and store id of identified state into temporary variable.
    // If condition part
    Identify and **output** the name of dependency having value of temporary variable as source id.
    Output the name of target (class) of identified dependency as well as attributes' names and their types of the target.
    // Then part
    Identify Then-labeled transition having value of temporary variable as source id.
    Output name of target state of the identified transition labeled with Then.
    // Else part
    Identify Else-labeled transition having value of temporary variable as source id.
    Output name of target state of the identified transition labeled with Else.

Step 2-5: **Case:** Repeat-While and Repeat-Until
Identify target state of Repeat-While (Repeat-Until) transition, and store id of that state (composite state) into temporary variable.
Identify If-Then-Else transition having value of temporary variable as id.
// While (Until) condition part
Apply If part of If-Then-Else case in same manner.
// Service to be repeated
Identify Then-labeled transition having value of temporary variable as source id.
Output name of target state of the identified transition labeled with Then.
Step 3: **Output** appropriate closing elements and **terminate**.

4. Implementation

4.1 Simple Scenario

As a simple scenario, we chose and adapted the one introduced in [4], which introduced the semantic web service first. This is about travel service: Someone wants to travel from one place to another via airplane or automobile. If the driving time from the departure place to the destination takes greater than 3 hours, then he/she would want to fly. Otherwise, he/she would rent a car. Furthermore, we assume the entire travel service is composed of two separate steps: (1) deciding on the transportation means, and (2) selecting accommodation. There are two possible transportation means: airplane and automobile.

4.2 Design[4]

Fig. 8 depicts our scenario mentioned in the previous subsection. We used the class diagram to logically model the travel service composed of three atomic services: *AirlineTicketing*, *CarRental* and *HotelReservation*.

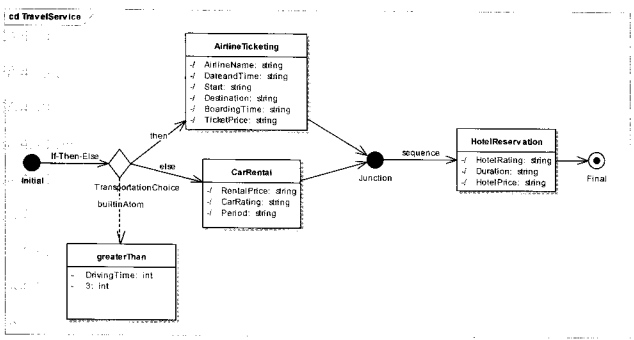


Fig. 8. Class diagram for travel service

[4] Note that extra labeling process is not needed when existing UML diagrams are already labeled during the service development process. This means the service ontology generation process can be fully automatic. Otherwise, developers just import UML profile for OWL-S and label onto UML diagrams being created. This means simplicity and effectiveness.

Fig. 9 describes that the entire service is composed of one composite service (*Transportation*) and atomic service (*HotelReservation*). It also depicts that the entire service must be executed in sequence. Fig. 10 is an expansion of the composite service *Transportation*. Note how the condition assumed in our scenario is modeled with UML primitives. The condition GUI indicates if driving time is greater than 3 hours (can be expressed as `greater than` (*DrivingTime*, *3*)), and then *AirlineTicketing* service would be used, otherwise *CarRental service*.

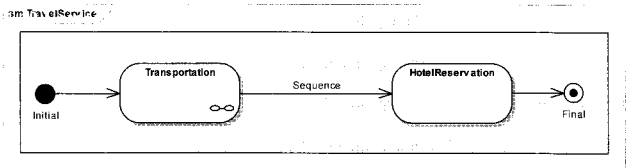


Fig. 9. State-chart diagram for entire travel (composite) service

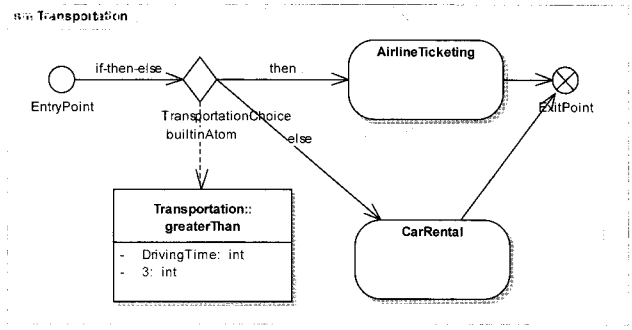


Fig. 10. State-chart diagram for *Transportation* service

4.3 Transformation

The UML class diagram and state-chart diagram are exported to 2 separate XMI files, and then two separate XSLT applications (written with the help of [41]) in Fig. 11 and Fig. 12 to produce output files. Finally, as shown in Fig. 13, the generated OWL-S service model ontology is produced as an integrated file.

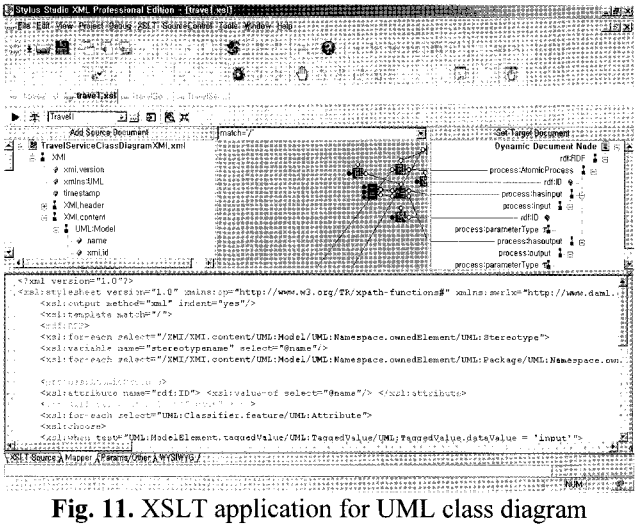


Fig. 11. XSLT application for UML class diagram

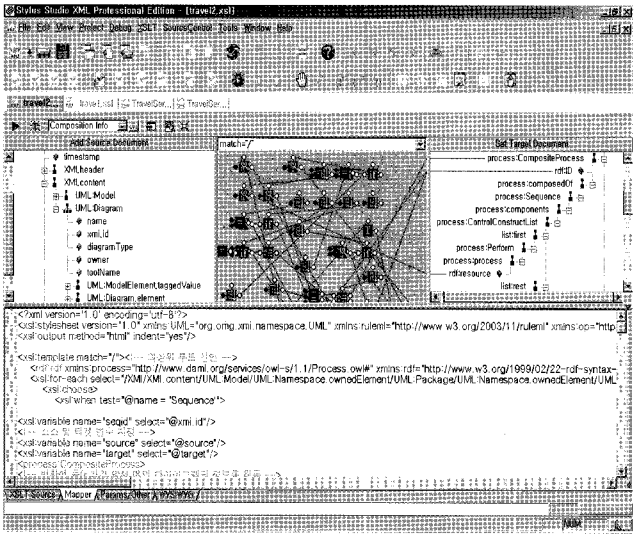


Fig. 12. XSLT application for UML state-chart diagram

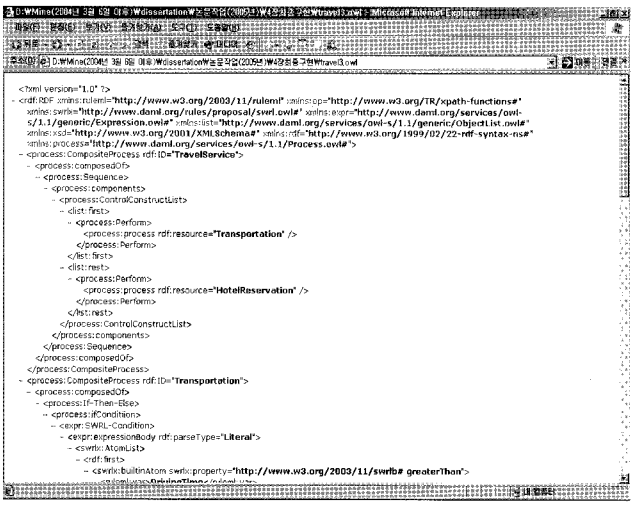


Fig. 13. Generated OWL-S service model ontology for travel scenario

4.4 Validation

We validated and confirmed the generated service ontology in several steps, since the OWL-S validator, provided by standard organization such as W3C, was not available. First, we used the site [33] available in W3C for the RDF-level test. Fig. 14 shows a successful RDF test. As W3C does not support beyond RDF, we had to use other sites to validate our generated ontology. Fortunately, a couple of OWL validators were available. Among them, we used [34] and verified as shown in Fig. 15. With respect to OWL-S validation, we found [35]. However, they do not support OWL-S version 1.1. Moreover, their OWL-S validator does not support all the OWL-S constructs for composite service - they support only `Sequence`, `Unordered` and `Split`. Therefore, we validated and confirmed our generated ontology, with the exception of the `If-Then-Else` construct part.

We decided not to perform semantic evaluation of the generated ontology at this time due to the following two reasons, and we attributed this validation to future works.

OntoClean [43] is a unique approach towards the formal evaluation of ontologies. In order to make it work, it is required to manually annotate a given taxonomy of concepts with a set of meta-properties[44]. This fact violates our principles of simplicity and automation.

And OntoClean methodology is not well suited for evaluating service ontology[5]. In order to ensure the correctness of the generated ontology, actual service referred by the ontology be tested, and then the results be checked against the intended ones. Real application use of the generated service ontology can be semantic matchmaking, which is another hot topic in the research field of semantic web services. To show the use of the generated service ontology (i.e., to verify the intended semantics of the ontology) is beyond this paper's topic.

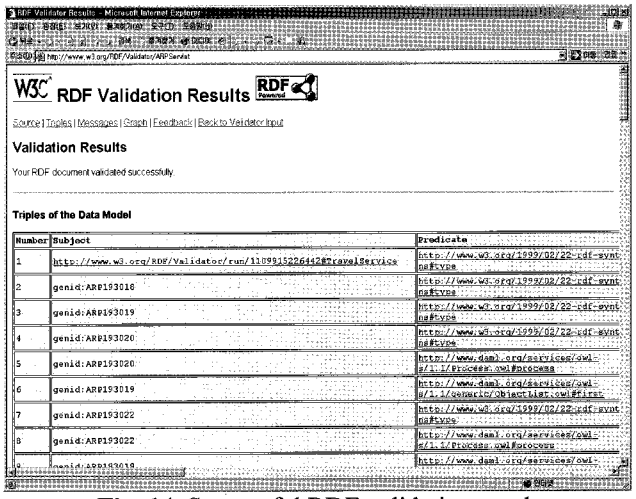


Fig. 14. Successful RDF validation result

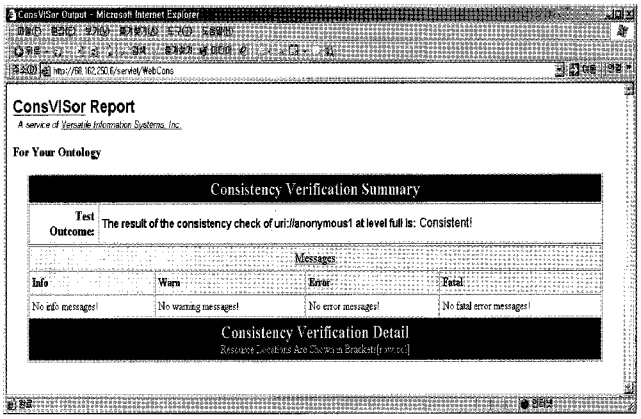


Fig. 15. Successful OWL validator result

5. Analysis and Discussion

In this section, we address and discuss several issues with respect to generating service ontology.

[5] Note that as we already mentioned in subsection 2.2, there exist differences between domain ontology and service ontology.

5.1 State-chart Diagram vs. Activity Diagram

We adopted a UML state-chart diagram to model service's composition. As a matter of fact, use of the state-chart or activity diagram for modeling service ontology is considered a future work in [20]. In practice, the activity diagram is used to model service's composition and generate BPEL4WS specification in [25].

The reason for using state-chart instead of the activity diagram is because the state-chart diagram not only has well-defined semantics, but also basic flow constructs such as sequence, conditional branching, structured loops, concurrent threads and synchronization primitives, as in most process modeling languages[36]. These features facilitate applying formal manipulation techniques to the state-chart model, and guarantee that state-chart can be adapted to other web service modeling languages such as BPEL4WS and WSCI[36].

5.2 Generality of Transformation Algorithm

Our transformation algorithm described in subsection 3.2 works not only in our scenario, but also in general cases. To explain this, we look first at the XMI tree structure exported from the UML state-chart.

Since all of OWL-S constructs for composite service are stored at UML:Transition elements in the XMI file, the algorithm works, in general, for any (composite) service composed of complex composite services.

We used EA [37] as the UML CASE and exported to XMI version 1.2. One thing noticeable is that our XSLT application may not work if a different UML CASE tool other than EA is used. This is due to the fact that XMIs exported from different UML CASE tools have different tree structures. In other words, EA, ArgoUML [38], and IBM Rational Rose [39] use their own methods (i.e., element naming) to generate XMI from the UML diagram. Even if we use the same UMAL CASE tool, a different XMI may be generated with a different XMI version.

In order to cope with the above issues, we considered possible solutions as follows: The XMI version issue is simple if we agree on using the same XMI version. Another problem arising from a different UML CASE tool forces one to build a new XSLT application. This is a critical problem, not because we need to create another XSLT application, but because different XMI files from the same model prevents sharing and reuse of the metadata (XMI). However, this kind of problem can be resolved if we define mappings between the two XMI files. Namely, we can create another XSLT that takes XMI from one UML CASE tool and transforms to the other XMI, which conforms to XMI from another UML CASE tool. In fact, these kinds of issues, as well as the possible solutions we recommended, are precisely mentioned in [40].

5.3 UML Profile

To maintain interoperability with UML specification, we created a UML profile where stereotyped transitions used in defining mappings in section 3 are described. Fig. 16 depicts the contents of the profile imported in the design phase. We used the *stateflow* type for OWL-S constructs for composite service, *dependency* type for three kinds of SWRL Atoms, and *class* type for instances of `built-inAtom`. Use of the *dependency* type rather than the *stateflow* type was due to the fact that the target type is not state, but class.

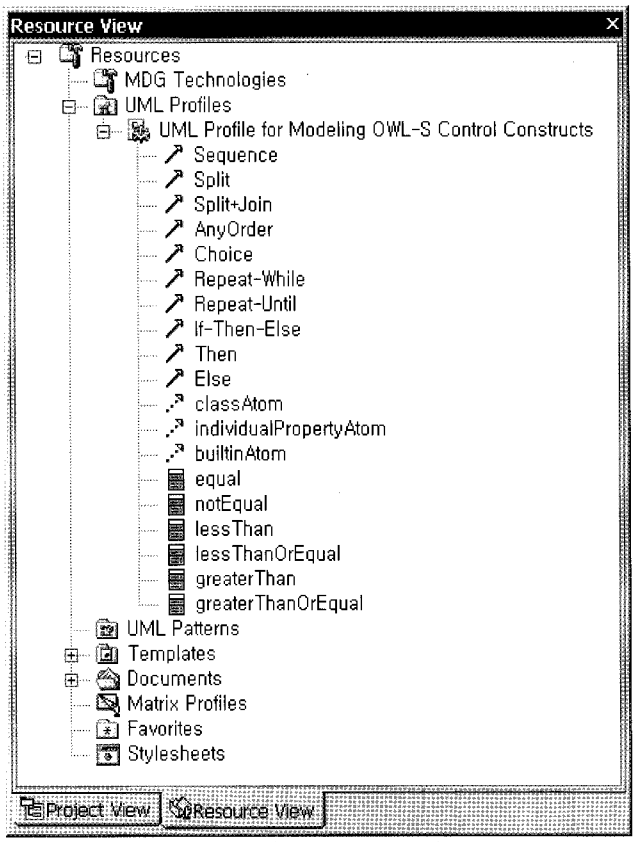


Fig. 16. UML profile for stereotyped OWL-S control constructs for composite services.

6. Conclusion and Future Works

We proposed a method for generating OWL-S service model ontology where service's behavior was described. First, we extracted information related to atomic services and their attributes, including IOPEs, from the UML class diagram UML, and extracted information related to service's composition from the UML state-chart diagram. Then, these two metadata were exported to two separate XMI files. Next, we generated the OWL-S service model ontology through XSLT applications based on the predefined mappings between UML diagram primitives and OWL-S constructs. Our approach for generating service ontology is general and fully automatic, as well as effective, in that it is performed in a familiar environment and information needed to generate service ontology is provided necessarily during service development. Another

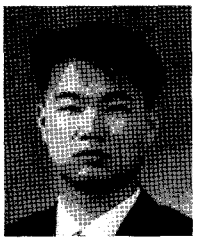
contribution may be representing condition using GUI rather than a complex language such as OCL. For the justification of generated ontology, we performed several available validation checks.

As future works, we considered the method of expressing with GUI complex condition where more than one predicate are represented. Simply, several *dependency* and *class* types can be used to express the complex condition. However, we may encounter a case where it is necessary to carefully consider the evaluation order of the individual condition of the complex condition expression.

References

[1] WSDL, http://www.w3.org/TR/2004/WD-wsdl20-primer-20041221/
[2] UDDI, http://www.uddi.org/
[3] SOAP, http://www.w3.org/TR/2003/REC-soap12-part0-20030624/
[4] Sheila A. Mcllraith, Tran Cao Son, Honglei Zeng, Semantic Web Services, IEEE Intelligent Systems, pp.46-53, 2001.
[5] BPEL, http://www-128.ibm.com/developerworks/library/specification/ws-bpel/
[6] WSCI, http://www.w3.org/TR/wsci/
[7] OWL-S, http://www.daml.org/services/owl-s/1.1/
[8] OWL, http://www.w3.org/TR/owl-features/
[9] http://www.daml.org/services/owl-s/1.1/related.html
[10] http://www.daml.org/services/daml-s/0.9/survey.pdf
[11] Michele Missikoff, Roberto Navigli, Paola Velardi, The Usable Ontology: An Environment for Building and Assessing a Domain Ontology, ISWC 2002, LNCS 2342, pp.39-53, 2002.
[12] Vijayan Sugumaran, Veda C. Storey, Ontologies for Conceptual Modeling: Their Creation, Use, and Management, Data & Knowledge Engineering 42, pp.251-271, 2002.
[13] Melania D., Vasileios H., Building Automatically a Business Registration Ontology, Proceedings of the 2002 National Conference on Digital Government Research, 2002.
[14] Patrick C., Padraig C., Conor H., Ontology Discovery for the Semantic Web using Hierarchical Clustering, Trinity College Dublin Computer Science Department, Technical Reposts, 2001.
[15] Armin W., Oliver W., Josef M. Joller, Siu Cheung Hui, Data Mining for for Ontology Building, IEEE Intelligent Systems, 2003.
[16] Mario C., Carmela C., A Data Mining Ontology for Grid Programming, 1st Workshop on Semantic in P2P and Grid Computing at the 12th Internal WWW Conference, 2003.
[17] Cranefield S., Purvis M., UML as a Ontology Modeling Language, Proc. Of the Workshop on Intelligent Information Integration, 16th Int. Joint Conference on AI(IJCAI-99), 1999.
[18] Cranefield, S., Haustein, S., and Purvis, M., UML-Based Ontology Modelling for Software Agents, Proceedings of the Workshop on Ontologies in Agent Systems, 5th Internal Conference on Autonomous Agents, pp.21-28, 2001.
[19] K. Baclawski, M. Kokar, P. Kogut, L. Hart, J. Smith, W. Holmes, J. Letkowski, M. Aronson, P. Emery, Extending the UML for Ontology Development, SOSYM 2002, Software System Model(2002) Vol.1, pp.1-15, 2002.
[20] K. Falkovych, M. Sabou, H. Stuchenschmidt, UML for the Semantic Web: Transformation-Based Approaches, Knowledge Transformation for the Semantic Web, IOS Press, pp.92-106, 2003.
[21] D. Duric, D. Gasevic, V. Devdzic, A MDA-based Approach to the Ontology Definition Metamodel, In Proc. Of the 6th International Conference on Information Technology, pp.193-196, 2003.
[22] Gannod C., Timm J., An MDA-Based Approach for Facilitating Adoption of Semantic Web Service Technology, In Proc. Of the 8th IEEE Enterprise Distributed Object Computing Conference Workshop on Model-Driven Semantic Web, 2004.
[23] P. Kogut, S. Cranefield, L. Hart, M. Dutra, K. Baclawski, M. Kokar, J. Smith, UML for Ontology Development, Knowledge Engineering Review Journal Special Issue on Ontologies in Agent Systems Vol.17, 2002.
[24] A. H., E. J., and Nicholas K., ASSAM: A Tool for Semi-automatically Annotating Semantic Web Services, ISWC 2004, LNCS 3298, pp. 320-334, 2004.
[25] Keith Mantell, From UML to BPEL: Model Driven Architecture in a Web Services world, http://www-128.ibm.com/developerworks/webservices/library/ws-uml2bpel/
[26] Protégé, http://protege.stanford.edu
[27] http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/OwlSEdit.html
[28] SWRL, http://www.daml.org/2004/04/swrl/
[29] RDF, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/
[30] Knowledge Interchange Format: Draft proposed American National Standard(dpans). Technical Report 2/98-004, ANS, 1998.
[31] M.Ghallab et al., Technical Report, report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.
[32] RuleML, http://www.ruleml.org/
[33] RDF validator, http://www.w3.org/RDF/Validator/
[34] ConsVISor, http://www.vistology.com/consvisor/
[35] http://www.mindswap.org/2004/owls/ validator/
[36] Lianzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, Quan Z. Sheng, Quality Driven Web Services Composition, WWW2003, pp.411-421, 2003.
[37] EA, http://www.sparxsystems.com.au/
[38] ArgoUML, http://argouml.tigris.org/
[39] http://www-306.ibm.com/software/ rational/

[40] Kovse J., Harder T., Generic XMI-Based UML Transformations, In Proc. 8th Int. Conf. on Object-Oriented Information Systems(OOIS'02), pp.192-198, 2002.
[41] Stylus Studio XML, http://www.stylusstudio.com/
[42] Massimo Paolucci, Naveen Srinivasan, Katia Sycara, Takuya Nishimura, Towards a Semantic Choreography of Web Services: form WSDL to DAML-S, In Proceedings of First Internal Conference on Web Services(ICWS'03), pp.22-26, 2003.
[43] N. Guarino and C. A. Welty, A Formal Ontology of Properties, In Knowledge Acquisition, Modeling and Management, pp.97-112, 200.
[44] J. Volker et al, Automatic Evaluation of Ontologies, ISWC 2005, LNCS(3279), pp.716-731, 2005.



Jin-Hyuk Yang
He received the B.S. and M.S. degrees in Computer Science Dpt. from Korea University in 1998 and 2000, respectively. He is now Ph.D. candidate student in Korea University. His research interests include semantic web, ontology, web services and ubiquitous computing.

In-Jeong Chung
He received the B.S. and M.S. degrees in Computer Science Dpt. from Seoul National University in 1978, and Korea Advanced Institute of Science and Technology in 1980, respectively. He received Ph.D. from University of Iowa in 1989. He is now professor in Korea University. His research interests are in the area of intelligent web services, semantic web, ontology engineering, home network and ubiquitous computing.