# From Ontology to Semantic Web Service via Model-Driven System Development

Ting-Huei Li and Cheng-Chia Chen
*Department of Computer Science, National Chengchi University*
*{g9211, chencc}@cs.nccu.edu.tw*

## ABSTRACT

*In this paper we propose a model-driven approach by which developers can make use of existing ontology knowledge to help construct a partial implementation of semantic web services on related domains. The idea is to use existing formal ontologies for an application domain as the basis of requirement and system analysis. Our system then transforms these ontologies into platform neutral models conforming to the EMF metamodel, which, after additional refinements, can be used to construct platform dependent web service models as well as a partial implementation by following the typical MDA process with supporting tools. Finally, developers need only fulfill the lacked service logic and a complete web service can be obtained. In addition to providing and integrating supporting tools, the main contribution of our system is to leverage ontology for rapid construction of software systems and, especially, semantic web services.*

***Keywords:*** *semantic web service, model-driven system development, web service, Ontology, MDA.*

## 1 Introduction

Semantic web is now a hot research topic in the field of web technology. There have been more and more domain knowledge formalized in so-called ontology expressed by XML-based languages such as RDF [1], RDFS [2], and OWL [3] etc. Ontology manifests its applications in web development, making knowledge sharable on the web. It enables resources on the web to be retrieved by semantic contents instead of traditional keyword search, which we have been using since the inception of the world-wide web. Furthermore, software agents can be designed to understand the ontology knowledge and react automatically according to the retrieved ontology without the need of human intervention. Although emerging from the field of semantic web, there is no reason that ontology could not be applied to other fields. This motivates our interest in applying ontology to software development.

Recently, there is a new software development process called Model Driven Architecture (MDA) [4]. In MDA, software development is focused on creating models rather than writing program codes. The primary goals of MDA are portability, interoperability and reusability through an architectural separation of concerns between specification and implementation of software.

A web service is a software component designed to provide specific functionality to other applications over the internet. Currently, it uses Web Service Description Language (WSDL) [6] to specify its interface to access clients. This language does not have the ability to describe the functionality of web services, because it lacks semantic constructs. To enable the semantic contents of a web service to be

exposed and published, an extension of web service called semantic web service is proposed[9]. It extends web service by using web service ontology[9]. This ontology provides language constructs for describing the properties and capabilities of a web service and it thus enable better search, discovery, selection, composition, and integration of web services. Unfortunately it is complex and tedious for a general developer to write these complicated descriptions.

In this paper we propose an approach to develop semantic web service from existing ontology knowledge via model-driven system development. Given the goal of developing a semantic web service, we as developers need to prepare for formal ontologies related to the intended web service, which can either be collected from the web or developed by the service developers. After all ontologies are available, we transform them into a platform neutral model compliant with the EMF-supported Ecore metamodel [8] using EODM[11] in the EMF perspective of Eclipse platform. The developers can then refine the generated model by using EMF editor to generate a platform independent model (PIM). Next, the PIM should be annotated with additional information relevant with the requirement of a semantic web service to form a platform specific web service model (PSM) targeted at the AXIS[12] web service container. Finally a partial implementation of the service can be generated from the PSM by code generation. The partial implementation includes java interfaces, data model API, and service description files. Developers get most of their code generated from their service models. In addition, the complicated descriptions required for the deployment and publishing of a semantic web service are also generated. What remains to be done is the lacked business logic of the service which is usually not specified in the models.

# 2 Backgrounds

This section describes some backgrounds and technologies about ontology, web service, and MDA.

## 2.1 Ontology and semantic web services

An ontology is a set of concepts together with a set of properties and relationships between them. Ontologies make it possible to describe resources formally. It thus plays an important role in the architecture of semantic web, enabling web-based knowledge processing, sharing and reuse between applications.

The Web Ontology Language (OWL) [3] is an XML-based language for describing ontologies. It is based on RDF/RDFS[1,2] and provides a language for the description of semantic contents of resources on the web. In OWL a resources means anything on the web such as a web page or a service. Not only enabling content-based search, ontology can also be interpreted automatically by software agents. The ecommerce can thus benefits much from utilization of ontology.

Web services are loosely coupled software components which support machine to machine interaction on the web. These services defined their interfaces with a description file called Web Service Description Language (WSDL) [6]. WSDL is an XML-based language for describing the interface of the service, including message formats, URI address bindings, and transfer protocol used in the service. The service providers usually need to register and publish their service with some registry service such as Universal Description, Discovery, and Integration (UDDI)[7]. A typical scenario of web service invocation is shown in Figure 1.

When clients or agents need a service, they will query the UDDI registry to find an available and qualified web service, which would have been published to the UDDI registry by its provider. After receiving the available service information from a

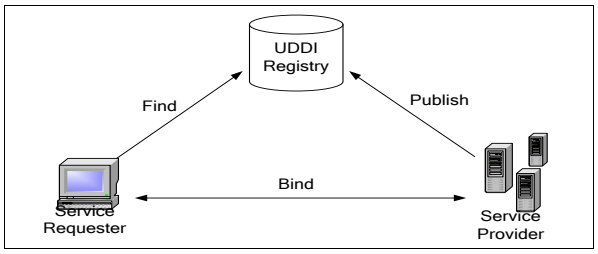


**Figure 1. Web Service Invocation**

registry, service requesters can retrieve the WSDL content associated with the service to create a binding to the service and then execute the service accordingly.

In the current state of web service, WSDL provides only I/O type and physical connection information for a web service, but it lacks semantic contents of the service it describes. In order to enable further applications of web service, it is necessary to enhance a web service by providing additional semantic information. This motivates the definition of OWL-S [9], which is an ontology for describing web services. It provides a formal mechanism for describing the semantics of a web service. Users and software agents can discover, select, compose, deploy, and monitor web services automatically by interpreting the associated semantic descriptions expressed in OWL-S.

There are three main parts in OWL-S ontology structure (shown in Figure 2): the service profile for advertising and discovering services; the process model giving a detailed description of a service's operation; and the service grounding providing details on how to interoperate with a service, via messages.

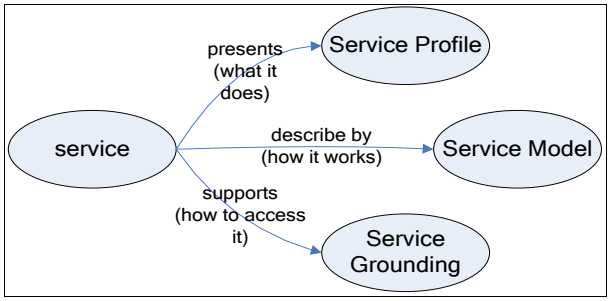


**Figure 2. OWL-S Structure**

## 2.2 MDA and EMF

Model Driven Architecture (MDA) [4] is a new way to develop software applications initiated by OMG. In MDA, system design is based on a Platform Independent Model (PIM) of the application or business functionality or behavior. The PIM can be translated to Platform Specific Models (PSM) with respect to an interface definition which describes how base model is implemented on a different platform. Therefore the portability, interoperability and reusability of system are increased through the separation of concerns between specification (PIM) and implementation of software (PSM).

Eclipse Modeling Framework (EMF) [8] is a modeling framework and code generation facility for building tools and applications based on structured data model in Eclipse Environment. The core EMF framework includes a metamodel (Ecore) for describing models, runtime support with default XMI serialization and reflective API for manipulating EMF objects. The Ecore metamodel is essentially a variant of EMOF[19] and similar to the subset of UML for modeling classes and their associations. Figure 3 shows a simplified subset of Ecore. It illustrates the relations between the four most important metaclasses: EClass, EAttribute, EDataType and EReference, of Ecore. EClass and EDataType are used to model managed and unmanaged java types, respectively; while EAttribute and EReference are used to model attributes and association ends of managed classes, respectively.

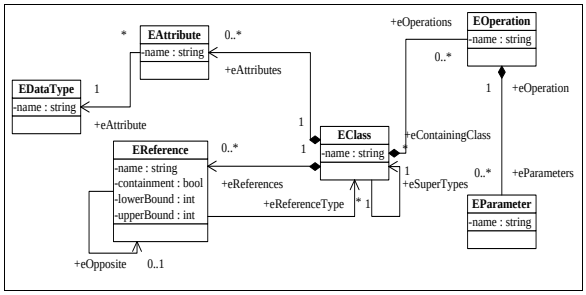


**Figure 3. Ecore Kernel**

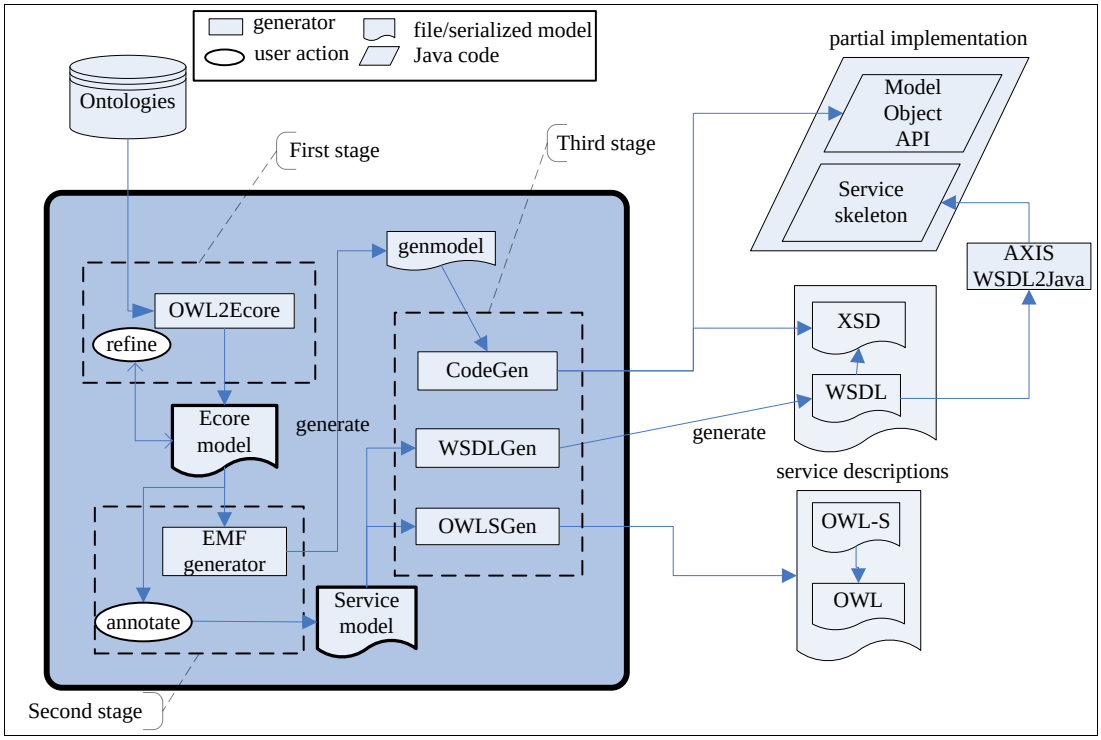


**Figure 4. System Architecture and Process**

Moreover, there are also EPackage and EAnnotation that would be used later. EPackage is for containment of EClasses and EAnnotation is for attaching additional information to model elements. Essentially, an EAnnotation is composed of one or more key-value pairs and has a URI identifying its source.

## 3 The process of our approach

This section describes the process of our approach to generate a semantic web service using existing ontologies. As shown in Figure 4, our semantic web service development process is divided into three stages.

The whole process starts with a set of domain ontologies collected in advance. These ontologies are specified by domain experts and are assumed to be stored in either OWL or RDF format. In the first stage, we import these ontologies into our system and transform them into a Ecore model. Since the model obtained this way usually does not meet the requirement of the intended web service, we need to manually refine this model by an editor and possibly merge it with other elaborated models to form a complete model. This model is a PIM (platform independent model) according to MDA since it is not only logically complete but also independent of the platform where it will be executed. In the second stage, we need to annotate this model with additional service-related information and then generate a platform specific service model targeted at the AXIS [12] web service container. In the final stage, we trigger the code generation procedure to generate a partial implementation of the service. The implementation includes Java source code and two description files required for a semantic web service. The generated Java source code includes model objects API and Java interfaces for the service; the description files include semantic descriptions (OWL-S) and concrete service description (WSDL). Figure 5 illustrates how these descriptions are used by service providers and requesters. Readers can refer to [9] for more details. These generated artifacts reduce most work a programmer would need to do while developing a web service. The three stages are detailed in the rest of this section.

## 3.1 From Ontology to PIM

To transform ontologies into models, we use the open source tool EODM [11] to do the transformation. It is an implementation of OMG's Ontology Definition Metamodel (ODM) specification. Besides other functionality, this tool can transform an ontology into a Ecore model in EMF. The critical part of the transformation is on the mapping rules between ontology and Ecore, which are briefed as follows:

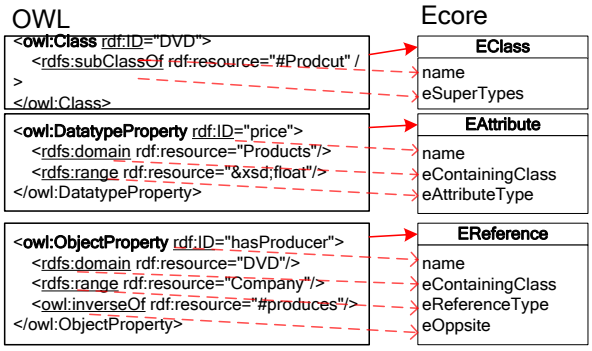


**Figure 5. Mapping between OWL and Ecore**

All other relationship descriptions between classes in OWL are mostly transformed into EAnnotations attached to the associated EClasses. After the transformation an Ecore model is generated and, thanks to EMF, it is saved in standard XMI format.

It is almost always the case that the model transformed from ontologies does not meet the requirements of the intended web service. Therefore we need to refine the model manually with some editing tools. Fortunately, the EMF framework, which our system works, has already provided for a friendly tree-structured editor for Ecore models. The developer can use it to edit the rough model and, if necessary, to import also other related Ecore models and merge them with it to form a complete platform independent model.

For instance, the model generated from ontologies should contain only static information such as structural features and class hierarchy of modeled entities but lack dynamic behavior information since at present there is no behavior description in the original ontologies. We thus have to add into the model the behavioral part of the intended web service. In EMF, behavior of entities is modeled by one or more EOperations. Each (instance of) EOperation is used to model an object method or web service operation. Therefore, for each operation of a target class we should create an EOperation to model the operation and add the EOperation to the EClass corresponding to the target class.

## 3.2 From PIM to PSM

This stage aims to decorate the PIM we obtained at stage 1 with additional information related to semantic web service so that finally we can generate a separate one which contains all required information of a semantic web service. The process of this stage is shown in Figure 6 and the resulting service model is the so-called PSM in terms of MDA.

The PIM we got from stage 1 contains many EClasses in one EPackage. It is certain that all EClasses are not to be semantic web services and for each EClass selected as a service there should be only part of its operations exposed as a service operation.

To inform later code generator of our selections of EClasses and EOperations as services and operations, respectively, we attach an EAnnotation to each selected one. Every EAnnotation has a same URI designated by us for identification; for an EOperation selected as a service, the attached EAnnotation has a 'gen' key with 'true' value. There are some other annotations which can be used to customize related element settings in generated WSDL and OWL-S.

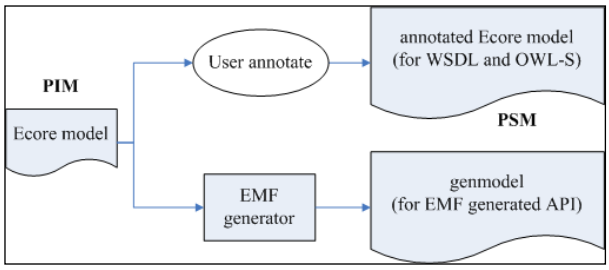


**Figure 6. The process in stage 2**

These models can be used in next stage to generate description files and partial implementation. The models can also be serialized in XMI formats for further uses. This provides reusability and portability to our service design.

## 3.3 From PSM to implementation code

In this stage, we designed two generators to generate WSDL and OWL-S descriptions from Ecore models. We then use Axis tool WSDL2Java to generate service stubs. We also generate model object API using EMF generator. At last, we integrate the service stub and object API to construct a complete web service implementation. The process of this stage is shown in figure 7.

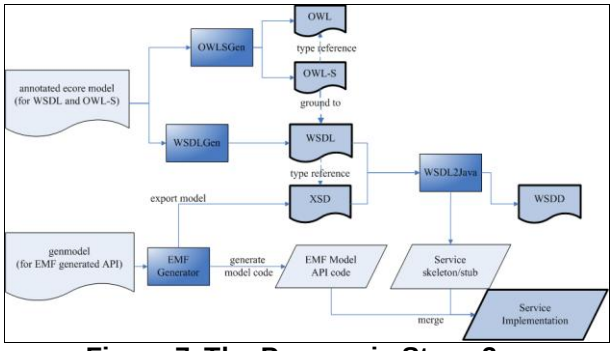


**Figure 7. The Process in Stage 3**

Ecore2WSDL and Ecore2OWLS module take the model structure data as its input and generate WSDL and OWL-S files. The mappings between Ecore and generated descriptions are listed below:

**WSDL:** WSDL is about concrete connection of a web service. Every EOperation is mapped to a *port/port type*; every EParameter is mapped to a *message/message part* in WSDL. In this WSDL we import the XSD type definition generated by EMF. So we don't have to redefine the message types in WSDL.

**OWL-S:** OWL-S is about semantics of a web service. In OWL-S, a service can only contain one service (atomic or composite). Therefore in our transformation, every EOperation will generate one .owl file that represents an atomic process, and all related input/output parameters will be generated from EParameter and return type in EOperation. In the service grounding part of OWL-S all grounding information between WSDL and OWL-S will be set automatically, because these two descriptions is generated from the same model.

After generate the descriptions, we can use AXIS [12] to develop our service. Because the WSDL files are already generated we can develop the service using top-down design. AXIS supports to generate service stubs and Javabean API for every I/O parameter from WSDL files. In order to benefit more from EMF, we substitute the original javabean API with EMF generated API. The EMF API has more powerful function such as reflection and XMI serialization. We also implemented the EMF serializer and deserializer for AXIS, so that we can transform EMF objects to SOAP message and vice versa.

When deploying the service to AXIS, in default setting, AXIS can only serialize java bean objects. We implemented a EMF serializer and deserializer using EMF reflective API and persistence API. After customize the WSDD settings in AXIS. The AXIS server can find serializer for the EMF objects and then transform them in to SOAP messages.

After the generation we can construct a service implementation, which only lack of logics in operations. In this manner, we don't have to write all service code. We only need to complete the logics in the service implementation.

# 4 Example

In this section we present an example to demonstrate our approach. In this example, we have an ontology about product and location information (a simple graphic view is shown in Figure 8). Our objective is to create two semantic web service operations called `getPrice` and `order` in a `NetStore` web service. The input and output types of both operations are an individual also from the example ontology.

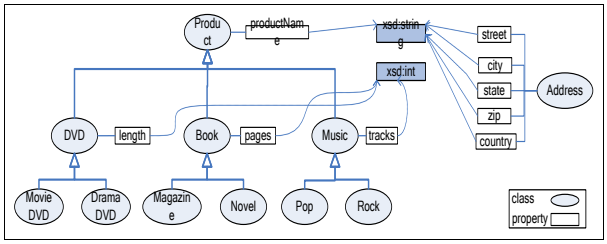


**Figure 8. Example Ontology**

This figure shows the subclass and instance relations between resources in the ontology. For example, the `Product` class has three subclasses and one attribute. This Product class is represented in Figure 6 using the OWL language.

```
<owl:Class rdf:ID ="&netstore;Product"/>
<owl:Class rdf:about="&netstore;DVD">
  <rdfs:subClassOf       rdf:resource="&netstore;Product"/>
</owl:Class>
<owl:Class rdf:about="&netstore;Book">
  <rdfs:subClassOf rdf:resource="&netstore;Product"/>
</owl:Class>
<owl:Class rdf:about="&netstore;Music">
  <rdfs:subClassOf rdf:resource="&netstore;Product"/>
</owl:Class>
<owl:DatatypeProperty rdf:about="&netstore;productName">
        <rdfs:domain rdf:resource="&netstore;Product"/>
        <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

**Figure 9. Example OWL ontology**

Now we step into the first stage of our approach: the generation of a PIM from ontology. First, the ontology will be imported and transformed into a Ecore model using EODM. However, at this time, the model contains no EClass for `NetStore`, our intended service, so we must manually add an EClass for `NetStore` into the model; meanwhile, two EOperations corresponding to `getPrice` and `order` must be added into the new EClass as well. We can also add more classes or attributes, if we need more. After the refinement we get a complete Ecore model, which is saved in XMI format.

While editing the two EOperations, since we have a data model of the ontology, it is easy to specify their return and parameter types. All available types will be shown in a drop down menu and are selectable by the user. This would prevent the user from putting invalid types in the type settings. The Ecore model we create in this example is shown in Figure 10.

In the second stage, we edit the Ecore model in the EMF model editor.

In the third stage, we generate description files from the service model using WSDLGen and OWLSGen modules, respectively. The generated WSDL and OWL-S from EOperation `getPrice` is shown in Listing 1, 2 and 3.

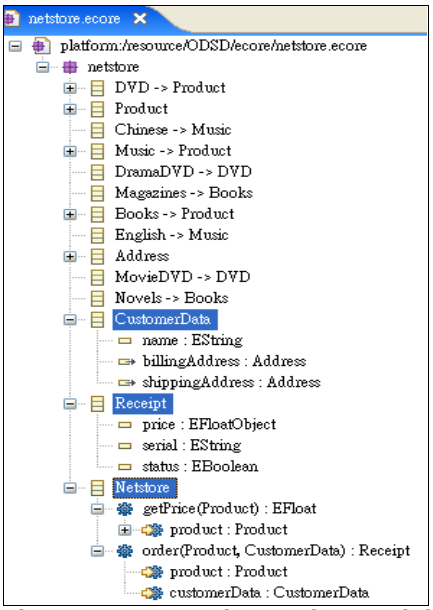


**Figure 10. Example Service Model**

# 5 Related works

There are some researches about transformation between ontology and models [13]. Most of these researches aimed to develop ontologies with MDA technology. They used UML profiles or extensions of UML as the metamodel of PIMs, and transform PIM to ontologies through XMI

```
    <wsdl:message name="getPriceInput">
      <wsdl:part name="product" type="tns:Product"/>
    </wsdl:message>
    <wsdl:message name="getPriceOutput">
      <wsdl:part name="getPriceOutput" type="xsd:float"/>
    </wsdl:message>
    <wsdl:portType name="NetstorePortType">
      <wsdl:operation name="getPrice">
        <wsdl:input message="typens:getPriceInput"/>
        <wsdl:output message="typens:getPriceOutput"/>
      </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="NetstoreBinding"
type="typens:NetstorePortType">
      <soap:binding style="document"
transport="http://schemas.xmlsoap.org/wsdl/soap/http/"/>
      <soap:binding style="document"
transport="http://schemas.xmlsoap.org/wsdl/soap/http/"/>
      <wsdl:operation name="getPrice">
        <soap:operation
soapAction="http://owl.cs.nccu.edu.tw/getPrice"/>
        <wsdl:input>
          <soap:body parts="product" use="literal"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://owl.cs.nccu.edu.tw/netstore/"/>
        </wsdl:input>
        <wsdl:output>
          <soap:body parts="getPriceOutput" use="literal"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://owl.cs.nccu.edu.tw/netstore/"/>
        </wsdl:output>
      </wsdl:operation>
      </wsdl:binding>
    <wsdl:service name="netstoreService">
      <wsdl:port name="NetstorePort"
binding="typens:NetstoreBinding">
        <soap:address
location="http://localhost:8080/axis/services/NetstorePort"/>
      </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

**Listing 1. Generated WSDL**

and XSLT technology. This allows users to develop ontologies with existing visual UML tools. Then, as the demands for the standardization of ontology modeling are increasing, OMG began to initiate the Ontology Definition Metamodel [14] and now it is in nearly complete state. It provides mainly a metamodel for ontologies languages and defines the mapping between conforming models and several ontology. Our approach uses the EODM implementation of ODM in the first stage.

In the research area of semantic web service development, there were some works about creating OWL-S descriptions. In [15], it proposed a UML Profile for OWL-S ontology, helped users to create OWL-S files in UML tools. The work in [16] used a template for OWL-S and a matchmaker system to select suitable

```
<service:Service rdf:ID="NetstoreGetPriceService">
      <service:presents>
        <profile:Profile rdf:ID="NetstoreGetPriceProfile"/>
      </service:presents>
      <service:supports>
        <grounding:WsdlGrounding>
          <grounding:hasAtomicProcessGrounding>
            <grounding:WsdlAtomicProcessGrounding
rdf:ID="NetstoreGetPriceGrounding"/>
          </grounding:hasAtomicProcessGrounding>
          <service:supportedBy
rdf:resource="#NetstoreGetPriceService"/>
        </grounding:WsdlGrounding>
      </service:supports>
      <service:describedBy>
        <process:AtomicProcess
rdf:ID="NetstoreGetPriceProcess"/>
      </service:describedBy>
    </service:Service>
    <profile:Profile rdf:about="#NetstoreGetPriceProfile">
      <service:presentedBy
rdf:resource="#NetstoreGetPriceService"/>
      <profile:hasOutput>
        <process:Output rdf:ID="getPriceResult">
          <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://www.w3.org/2001/XMLSchema#float</process:parameter
Type>
        </process:Output>
      </profile:hasOutput>
      <profile:hasInput>
        <process:Input rdf:ID="product">
          <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://owl.cs.nccu.edu.tw/Product</process:parameterType>
          <rdfs:label>product</rdfs:label>
        </process:Input>
      </profile:hasInput>
    </profile:Profile>
    <process:AtomicProcess
rdf:about="#NetstoreGetPriceProcess">
      <rdfs:label>NetstoreGetPriceProcess</rdfs:label>
      <process:hasOutput rdf:resource="#getPriceResult"/>
      <process:hasInput rdf:resource="#product"/>
      <service:describes
rdf:resource="#NetstoreGetPriceService"/>
    </process:AtomicProcess>
```

**Listing 2. Generated OWL-S profile and process**

ontology into the template and then generate OWL-S from it. In [17] and [18], they proposed an MDA-based approach to create and composite OWL-S descriptions. They used WSDL files to generate OWL-S groundings in a semi-automatic tool and to compose a composite semantic web service in UML activity diagram, and then generated OWL-S with XSLT.

All above works are concerned with the composition of composite web services from existing ones. Our approach, on the other hand, is about how to build an atomic web service with its initial model derived from existing ontology knowledge. This

```
<grounding:WsdlAtomicProcessGrounding
rdf:about="#NetstoreGetPriceGrounding">
    <grounding:wsdlOutputMessage
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://owl.cs.nccu.edu.tw/netstore/netstore.wsdl#getPrice
    </grounding:wsdlOutputMessage>
    <grounding:wsdlInputMessage
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://owl.cs.nccu.edu.tw/netstore/netstore.wsdl#getPriceInput
    </grounding:wsdlInputMessage>
    <grounding:owlsProcess
rdf:resource="#NetstoreGetPriceProcess"/>
    <grounding:wsdlInput>
      <grounding:WsdlInputMessageMap>
        <grounding:owlsParameter rdf:resource="#product"/>
          <grounding:wsdlMessagePart
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://owl.cs.nccu.edu.tw/netstore/netstore.wsdl#product
          </grounding:wsdlMessagePart>
      </grounding:WsdlInputMessageMap>
    </grounding:wsdlInput>
    <grounding:wsdlOperation>
      <grounding:WsdlOperationRef>
          <grounding:portType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://owl.cs.nccu.edu.tw/netstore/netstore.wsdl#getPricePortTyp
e
          </grounding:portType>
          <grounding:operation
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://owl.cs.nccu.edu.tw/netstore/netstore.wsdl#getPrice
          </grounding:operation>
      </grounding:WsdlOperationRef>
    </grounding:wsdlOperation>
    <grounding:wsdlDocument
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://owl.cs.nccu.edu.tw/netstore/netstore.wsdl
    </grounding:wsdlDocument>
    <grounding:wsdlOutput>
      <grounding:WsdlOutputMessageMap>
          <grounding:wsdlMessagePart
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://owl.cs.nccu.edu.tw/netstore/netstore.wsdl#getPriceOutput
          </grounding:wsdlMessagePart>
        <grounding:owlsParameter
rdf:resource="#getPriceResult"/>
      </grounding:WsdlOutputMessageMap>
```

**Listing 3. OWL-S Groundings**

approach allows us to create an atomic service easier and faster. The created services can be deployed and published without dependency on other services, and the clients can either execute them directly or compose them into parts of other composite services.

## 6 Conclusions

Semantic web service is a new technology in web development. But the service ontology is complex and verbose, so it is hard for common developer to create such ontologies. We propose an approach to transform existing ontology knowledge into neutral models and using these models to build service models. We implemented the generators to generate these service descriptions from service model. So developers only need to create the service model, and then service description files and a partial implementation can be generated automatically. Via this MDA-based approach, an atomic process can be built much easier and faster. Thus this approach not only provides a way to utilize existing ontology knowledge, but also makes it easier and faster for common developers to build a semantic web service.

## 7 References


[1] Brian McBride, Hewlett-Packard Laboratories, RDF Primer, http: //www.w3. org/TR/rdf-primer/

[2] Dan Brickley, RDF Vocabulary Description Language 1.0: RDF Schema, http://www.w3.org/TR/rdf-schema/

[3] Michael K. Smith, Chris Welty and Deborah L. McGuinness, OWL Web Ontology Guide, http://www.w3.org/TR/owl-guide/

[4] Joaquin Miller and Jishnu Mukerji et al. MDA Guide Version 1.0.1 Technical Report omg/2003-06-01, http://www.omg.org/docs/omg/03-06-01.pdf

[5] Anneke Kleppe, Jos Warmer, Wim Bast, MDA Explained: The Model Driven Architecture: Practice and Promise, Addison Wesley

[6] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001, http://www.w3.org/TR/wsdl

[7] Universal Description, Discovery, and Integration spec, http://www.oasis-open.org/ committees/uddi-spec

[8] Frank Budinsky, Ray Ellersick, Timothy J. Grose, Ed Merks, David Steinberg, Eclipse Modeling Framework, Addison Wesley

[9] OWL-S: Semantic Markup for Web Services, http://www.daml.org/services/owl-s/1.1/over view/

[10] OWL-S Informal Grounding Presentation, http://www.daml.org/services/owl-s/1.1/owl-s-wsdl.html

[11] EODM in Eclipse Modeling Framework Technology project, http://www.eclipse.org/ emft /projects/eodm/

[12] AXIS in Apache web services project, http://ws.apache.org/axis/
[13] Dragan Gašević, Vladan Devedžić, Dragan Djurić, MDA standards for ontology development, http://afrodita.rcub.bg.ac.yu/~gasevic/tutorials/ICWE2004/
[14] Ontology Definition Metamodel , Sixth Revised Submission to OMG/ RFP ad/2003-03-40, http://www.omg.org/cgi-bin/doc?ad/06-05-01.pdf
[15] Roy Grønmo, Michael C. Jaeger and Hjørdis Hoff, Transformation between UML and OWL-S
[16] Michael C. Jaeger, Lars Engel, and Kurt Geihs, A Methodology for Developing OWL-S descriptions.
[17] John T.E. Timm, Gerald C. Gannod, A Model-Driven Approach for Specifying Semantic Web Services, in Proceeding of the 2005 IEEE International Conference on Web Services, July 2005.
[18] Gerald C. Gannod, Raynette J Brodie, and John T.E Timm, An interactive approach for specifying OWL-S grounding, in Proceedings of the 2005 IEEE EDOC Enterprise Computing Conference.
[19] OMG Inc., Meta Object Facility (MOF) Core Specification OMG Available Specification Version 2.0. 2006. http://www.omg.org/docs/formal/06-01-01.pdf