Using Semantic Web Technologies to Integrate Software Components

Dr. Marwan Sabbouh
Dr. Joseph K. DeRosa

The MITRE Corporation
Bedford, MA 01730

Abstract. This paper illustrates how to integrate software components at a semantic level without the need for software development. An Ontology Management System is used as a tool to create a domain ontology as well as ontologies for common software components: a relational database and a web service. We generate the database ontology from the relational algebra and the web service ontology from the Web Services Description Language file. We manually input additional semantics through the Ontology Management System, and subject matter experts use it to link the Component Ontologies to the Domain Ontology. By using this methodology, we are able to automatically generated integration code from the linked ontology graph. Thus, in integrating new software components, we trade the work of subject matter experts for that of code developers. We illustrate the procedure with simple examples.

1 Introduction

This paper uses Semantic Web technologies [1-3] to integrate disparate software components – without the need for new software development. We capture the entities, relationships, and business rules of the information system (IS) in a domain ontology, as defined by Gruber [4] and Guarino [5]. We likewise build ontologies for software components that exist at any layer of an N-tier architecture, e.g., a relational database (RDB) at the Data Tier and a web service (WS) at the Web Tier. We build and link the domain and component ontologies through an ontology management system (OMS) that exposes those resources in a language familiar to the business users and subject matter experts (SMEs). The linked ontologies serve as semantic "glue" that integrates software components in the various tiers, and the OMS serves as the mechanism to identify integration paths, i.e., executable paths through the ontology graphs. Not only can users now access disparate software components through these integration paths, but also the system can automatically generate code following these paths. A new integration paradigm emerges:

1. Express any software component's formalism as an ontology,
2. Map the component's ontology to the domain ontology, and thus
3. Integrate components at the semantic level without writing integration code.

We have successfully demonstrated these techniques in several prototype applications.

For this analysis we consider the integration of a legacy RDB and a new WS. We postulate ontologies, for the information system and each of the components, consisting of RDF/OWL [6,7] triples and their corresponding graphs. The database ontology consists of a database upper (or at least mid-level) ontology coupled with the ontological representation of the database relations (tables). The upper ontology specifies the structure, algebra and constraints of a generic relational database [8]. The Web service ontology is similarly based on the Web Services Description Language (WSDL) file. [9]

We use existing Ontology Management System (OMS) tools [10,11] to create and integrate ontological elements. SME's map concepts from the component ontologies to the domain ontology, then the OMS augments the domain ontology with instance data according to those defined mappings. The OMS manages the ontology analogously to the way the RDBMS manages data for legacy applications. Paths through the augmented ontology graph represent integrated information.

We then implement a Semantic Viewer that, for a given request, generates executable paths satisfying the request. We present the details and give simple examples. We also present a comparison with OWL-S and traditional middleware approaches.

2 Integration Using Semantic Web Technologies

To perform a simple test of the theory, we used an RDB that held customer locations and a WS (outside the core information system) that placed locations on a map. The steps we took to integrate the two components are as follows:

1. Create a domain ontology
2. Create and link the RDB component ontology to the domain ontology
3. Create and link the WS component ontology to the domain ontology
4. Broker a user request to suggest executable paths through the augmented ontology
5. Manually view the result of implementing one of the executable paths, or
6. Automatically generate a web service to do so

The information system uses the terminology "Street," "City or Town," "State," "Postal ZIP" and has the concepts of "customer" and "location" (of customer). The mapping web service, in this case Mapquest [12], has the different terminology "Address or Intersection," "City," "State," "ZIP Code" and the concept of "Map."

2.1 The Domain Ontology

In our domain ontology **D** we define the RDF classes **Business**, **Customer**, **Name**, **Location**, **Street**, **CityOrTown**, **State** and **PostalZIP**, the OWL object properties sellsTo, doesBusinessAs, and residesAt , and the OWL datatype property hasA with domain rdf:Class and range datatype string.
D is shown in figure 1.

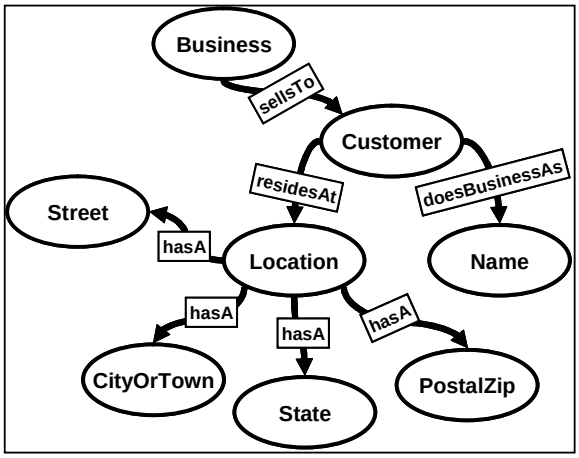

Figure 1 - Portion of the Domain Ontology

2.2 Integration of a Database into the Domain Ontology

Next we link concepts in the RDB ontology to concepts in the Domain Ontology. This not only allows us to automatically generate database queries based on the mappings, but also to classify the returned data as instances of the Domain Ontology. To do this we create a database component ontology. Note, it is not our intention to fully establish the OWL/RDF representation of the RDB relational algebra here, and we recognize that in some cases it will be a big task to represent a legacy RDB ontologically. However, we believe this can be mitigated by representing only portions of the RDB – i.e., those relational variables that map to the domain ontology.

This database ontology consists of a database upper (or at least mid-level) ontology coupled with the ontological representations of the database relations (tables) as instances of the upper ontology.

The upper ontology $\mathbf{R}^{\mathbf{U}}$ specifies the structure, algebra and constraints of any relational databases in RDF/OWL triples. We define the RDF classes **Database**, **Relation**, **Attribute**, **PrimaryKey** and **ForeignKey**. A portion of $\mathbf{R}^{\mathbf{U}}$ is given below:

$\mathbf{R}^{\mathbf{U}}$::{**Database**, hasRelation, **Relation**}
$\mathbf{R}^{\mathbf{U}}$::{**Relation**, hasAttribute, **Attribute**}
$\mathbf{R}^{\mathbf{U}}$::{**PrimaryKey**, subClassOf, **Attribute** }
$\mathbf{R}^{\mathbf{U}}$::{**ForeignKey**, subClassOf, **Attribute** }

where hasRelation and hasAttribute are OWL object properties, and subClassOf is defined in the RDF schema.

Consider the database relation **ADDRESS** with **Address_ID** as the primary key, and **Street**, **City**, **State**, and **Zip** as attributes. Two partial records are shown in Fig 2.

Address_ID	Street	Zip	--
001	255 North Rd.	01824	--
002	21 High St.	01851	--

Figure 2. Address Table

The ontological representation of Fig.2 is partly given by:

$\mathbf{R}^{L}$::{**Address**, isInstanceOf , **Relation**}
$\mathbf{R}^{L}$::{ **Address**, hasAttribute, **Address_ID** }
$\mathbf{R}^{L}$::{ **Address**, hasAttribute, **Street** }
$\mathbf{R}^{L}$::{ **Address**, hasAttribute, **Zip** }
$\mathbf{R}^{L}$::{**Street**, isInstanceOf, **Attribute** }
$\mathbf{R}^{L}$::{**Zip**, isInstanceOf, **Attribute** }
$\mathbf{R}^{L}$::{**Address_ID**, isInstanceOf, **PrimaryKey** }

where isInstanceOf specifies a class individual without inheriting the class properties (indicated in Figure 3 with ⇥).

The database ontology **R** is the conjuction of $\mathbf{R}^{L}$ and $\mathbf{R}^{U}$ as is shown in Figure 3.

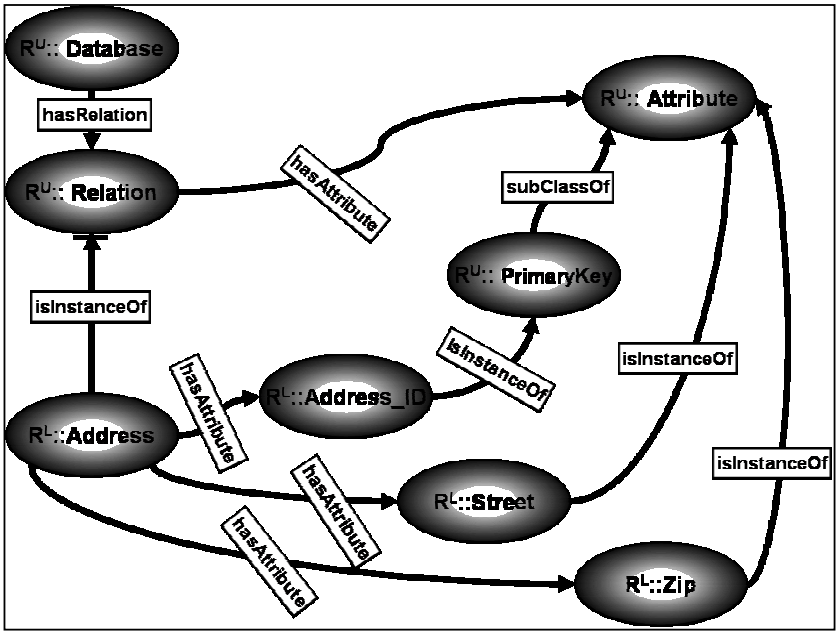

Figure 3. A Portion of the Database Ontology **R**

D$^+$::{ **D::Location**, hasSource, **R::Address**}
D$^+$::{ **D::Street**, hasSource, **R::Street**}
D$^+$::{ **D::CityOrTown**, hasSource, **R::City**}
D$^+$::{ **D::State**, hasSource, **R::State**}
D$^+$::{ **D::PostalZip**, hasSource, **R::Zip**}

A portion of the augmented Domain Ontology **D**$^+$ containing database instance data is shown in Figure 4.

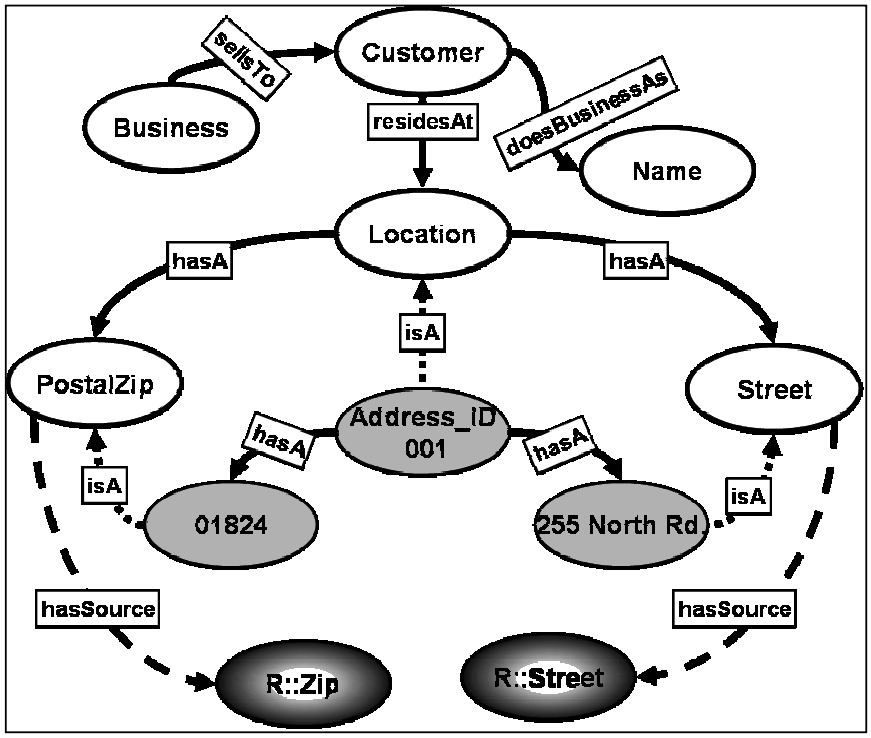

Figure 4. Augmented Domain Ontology **D**$^+$ Evaluated

At this point and in what follows, we drop the triple notation and revert to the equivalent graph notation.

2.3 Web Service Integration into the Domain Ontology D$^+$

To integrate a Web service into the IS, we again augment the domain ontology with concepts from the Web services description language (WSDL) file, and we create a WS component ontology that contains the invocation information of the Web service. Then, we link the WS ontology to the domain ontology.

A. The Web Service Upper Ontology $\boldsymbol{W^U}$

We postulate an upper ontology that models Web services as concepts of Inputs (inParameters), Output, Classification Conditions and Effects. Figure 5 shows a portion of the graph of $\mathbf{W^U}$. Classification Conditions are conditions that relate inputs to concepts in the domain ontology. These conditions must hold true in order that a Web service be able to operate on an instance of the domain ontology. Effects reflect property changes that are true as a result of running the Web service on instance data in the domain ontology. The Semantic Viewer (described below) makes use of both Classification Conditions and Effects. $\mathbf{W^U}$ is similar to the Service Profile of OWL-S.[13] (see Section 4.1).

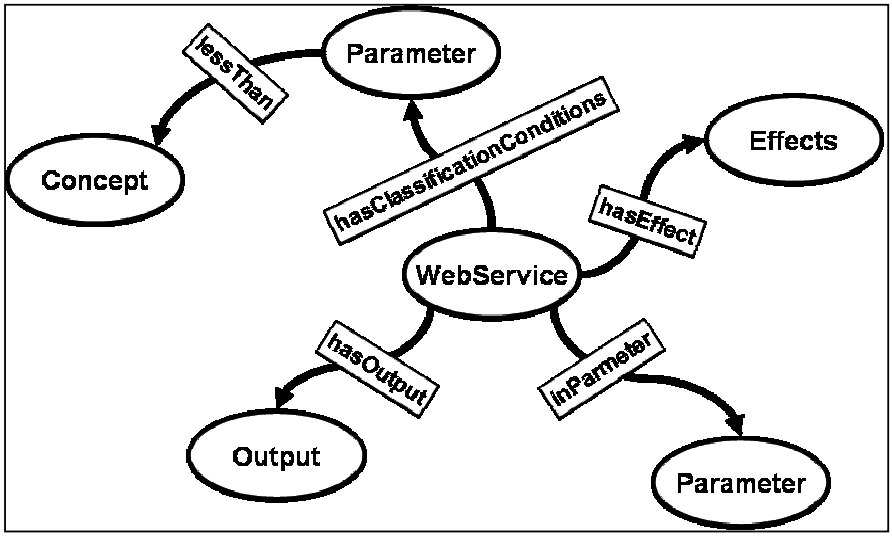

Figure 5. Portion of the Web Service Upper Ontology

B. Augmenting the domain ontology with $\boldsymbol{W^U}$ *to form Ontology* $\boldsymbol{D^{++}}$

When integrating a Web service into the IS, an instance of ontology $\mathbf{W^U}$ is created. The instance ontology $\mathbf{W^I}$, is then linked to the augmented domain ontology using the isInputTo relationship to form Ontology $\mathbf{D^{++}}$. Currently, this is done manually by the SME using the Edit service of the OMS. This is shown in Figure 6.

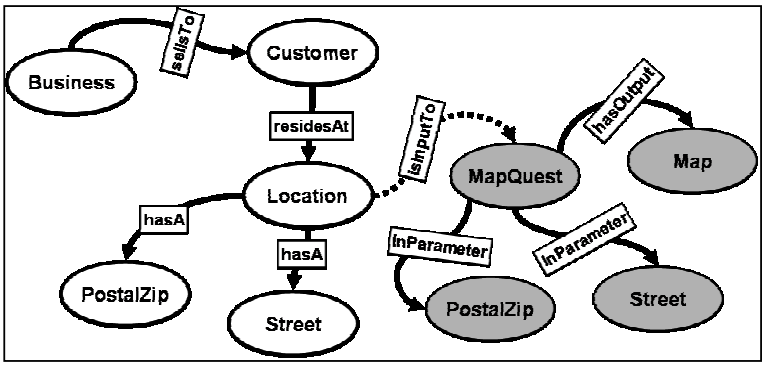

Figure 6. Instance Ontology $\mathbf{W^I}$

C. Web Service Invocation Upper Ontology $\boldsymbol{V^U}$

We created an upper ontology that models how to access and invoke a Web service. The ontology formalizes the WSDL specification [9] in RDF/OWL We have completed this upper ontology for REST-style web services [19] and have nearly completed it with the addition of SOAP-style web services. [20**]** **Figure 7** shows a subgraph of this ontology. $\mathbf{V^U}$ serves the same purpose as the Service Grounding of OWL-S [13] (see Section 4.1).

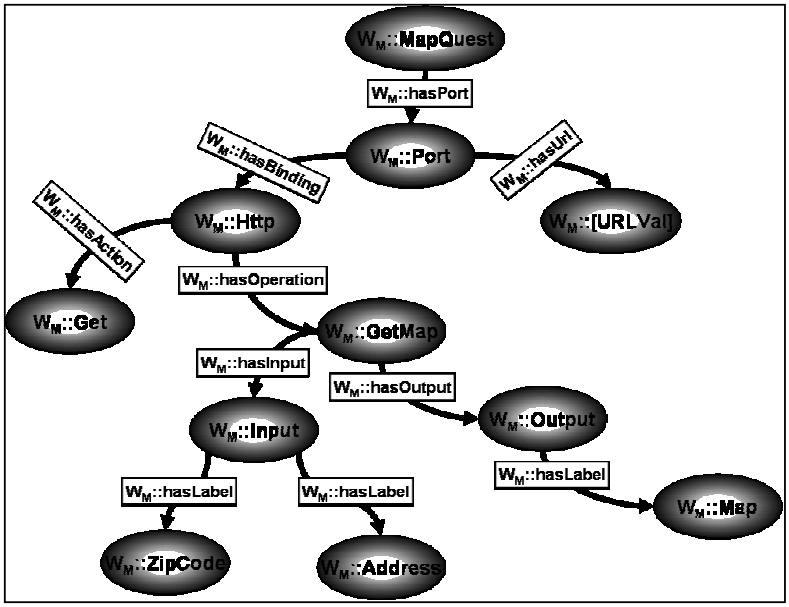

Figure 7. Mapquest Ontology

D. Web Service Ontology W_M : Instance of V^U

When integrating a web service into the ontology, the WSDL file is read by the WSDL MAPPER tool in the OMS, and a Web service ontology $\mathbf{W_M}$ is created.

The ontology $\mathbf{W_M}$ is an instance of $\mathbf{V^U}$. For our MapQuest example, the ontology is shown in Figure 7 (where [URL Val] =http://mapquest.com/ maps/map.adp).

E. Linking the Augmented Domain Ontology to Ontology W_M

In this step, the SME uses the isValueOf, isServedAt, and hasResult relationships to link the ontology $\mathbf{W_M}$ to the augmented domain ontology $\mathbf{D^{++}}$. The isValueOf links the range or object value of the hasLabel relationship in the ontology $\mathbf{W_M}$ to concepts in the augmented domain ontology $\mathbf{D^{++}}$. The isServedAt relationship links the subject or domain of the hasOutput relationships in the $\mathbf{D^{++}}$ ontology to the object of the hasUrl relationship in the $\mathbf{W_M}$. The hasResult relationship links the range of hasLabel relationship in the $\mathbf{W_M}$ to the range of hasOutput relationship in the ontology $\mathbf{D^{++}}$. This relationship is useful when the output of the Web service contains the inputs of another. This is shown in Figure 8 (where, for simplicity, $\mathbf{W^U}$ and $\mathbf{V^U}$ are not shown).

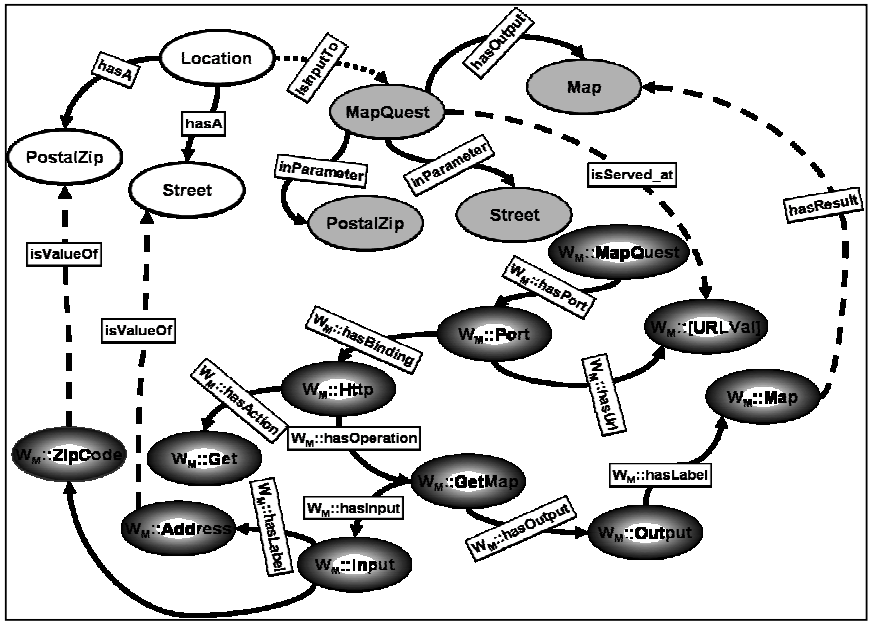

Figure 8. Example of Ontology and Instance Data in Graph Format

2.4 Broker a User Request to Suggest Executable Paths: The Semantic Viewer

To handle user requests, we create a Semantic Viewer that accesses $\mathbf{D^{++}}$ and returns a sub-graph of executable paths linking what the user specified as inputs and desired outputs. Reference [14] describes an algorithm to find such executable paths. Note an executable path now contains concepts and instance data from $\mathbf{D^{++}}$ (i.e., from $\mathbf{R^U}$,

$\mathbf{R}^{L}$, $\mathbf{W}^{U}$ and $\mathbf{W}^{I}$) that satisfy the user query. When a user clicks on an instance of a Web service in $\mathbf{D}^{++}$, the Semantic Viewer automatically invokes the Web service; automatically translates its output from XML to RDF; and finally classifies the RDF returned result as an instance in the domain ontology.

The Semantic Viewer application is created once and used many times in the application server layer (for all ontology-based integration of software components). We have built a Semantic Viewer as a web-enabled application. A user interacts with the Semantic Viewer through the userInput, userOutput, and userCriteria relationships. The userInput links the user's input to the object value of the inParameter relationship in the augmented domain ontology. The userOutput links a goal in which the user is interested to a concept in the augmented domain ontology. The userCriteria links user's input to concepts in the augmented domain ontology that are not input to web services. To illustrate the inner workings of the Semantic Viewer, we examine several scenarios:

CASE 1: Component Ontologies Have Matching Inputs/Outputs

When a user enters input1 = "01851", input2 = "255 North Rd." and output = "Map," i.e, the user requests a map, the SME links:

1. input1 to ZIPCODE using the userInput
2. input2 to STREET using the userInput
3. output to Map using the userOutput

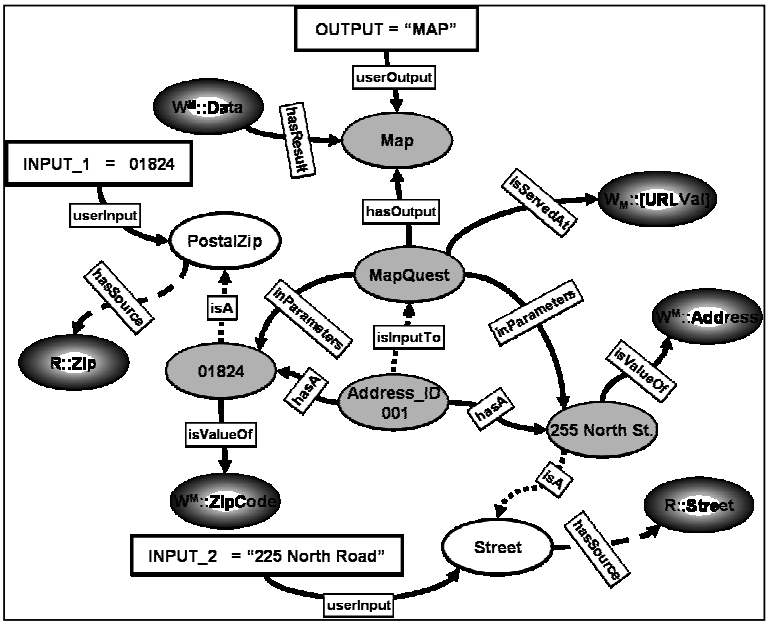

Figure 9. Executable Path

Having linked the request to concepts in the augmented domain ontology, and having found “01851”, “255 North Rd.” and “Map” in the augmented domain ontology $\mathbf{D}^{++}$, the Semantic Viewer does the following:

1. The Semantic Viewer issues the request of the Compose service
2. The Compose service selects a sub-graph from the ontology that connects “01851”, “255 North Rd.” with “Map,” as shown in Fig. 9.
3. The Semantic Viewer displays the sub-graph to the user
4. Upon a user’s mouse click, for example clicking map, the Viewer automatically invokes MapQuest.coml

CASE 2: Component Ontologies Do Not Have Matching Inputs/Outputs of Web Sevices

When a user enters input1 = “MITRE” and output = “Map,” i.e, the user requests a map for customer “MITRE”, the SME links:

1. input1 to NAME using the userCriteria
2. output to Map using the userOutput

In this case, it is clear that "NAME" is not an input to MapQuest. Having linked the request to concepts in the augmented domain ontology, and having found “MITRE”, in the augmented domain ontology D^{+}, the Semantic Viewer does the following:

3. The Semantic Viewer issues the request of the Compose service
4. The Compose service selects a sub-graph from the ontology that connects “MITRE”, with “Map,” and in the process retrieving the address of MITRE from the database as required by MapQuest.
5. The Semantic Viewer displays the sub-graph to the user
6. Upon a user’s mouse click, for example clicking map, the Viewer automatically invokes MapQuest.com passing it the ZIP Code and Street number found in the database associated with the name MITRE.

CASE 3: Classification Conditions and Multiple Execution Paths

In this case, we don't have a unique instance that the Web service can operate on as in the previous cases. This is where classification conditions become useful. A classification condition exists when the object of inParameter of a web service is linked to a concept in the domain ontology conditionally, (e.g., LessThan, GreaterThan,…) For instance, suppose a temperature weather forecast ontology exists for a region. i.e. temperature forecast for Northeast United States. Now suppose the existence of a Web service that for a given latitude and longitude, returns the temperature. In order for the Web service to operate on the temperature forecast, the latitude and longitude must fall within the Northeast United States region. In this case, the above mentioned service has in its $\mathbf{W}^{\mathbf{I}}$ ontology the following relationships:

{**Web Service**, hasClassificationConditions, **Latitude**},{ **Web Service**, hasClassificationConditions, **Longitude**},{**Longitude**, lessThan, **Longitude-**

East},{**Longitude**, greaterThan, **Longitude-West**},{**Latitude**, lessThan, **Latitude-North**},{**Latitude**, greaterThan, **Latitude-South**},{ **Web Service**, inParameter, **Longitude** },{ **Web Service**, inParameter, **Latitude** },{ **Web Service**, inParameter, **Forecast** },{**Forecast**, isInputTo, **Web Service** },{**Temperature**, isOutputTo, **Web Service** }

Thus, having not found latitude and longitude in the ontology $\mathbf{D}^{+}$, the semantic viewer does the following:

1. Queries for a Web service with inParameter = latitude and inParameter = llongitude, which returns the Web Service.
2. Queries for classification conditions on latitude and longitude, which returns Longitude-East, Longitude-West, Latitude-South, and Latitude-North.
3. Queries for the values of Longitude-East, Longitude-West, Latitude-South, and Latitude-North that satisfies the conditions: Longitude lessThan Longitude-East, Longitude greaterThan Longitude-West, Latitude lessThan Latitude-North, and Latitude greaterThan Latitude-South.
4. Passes the values in 3 along with the output temperature to the compose service.
5. Proceeds as in the previous case.

Case 4: Effects When There Is No Execution Path

Effects are useful for Web services that operate on instance data in order to produce other instance data. Suppose a temperature weather forecast is produced according to the Navy Operational Global Atmospheric Prediction (NOGAP) System. Assume a user desires a temperature weather forecast that is produced according to the Mesoscale Model (MM5) forecasting model. Now, suppose the existence of a Web service that operates on NOGAP and produces MM5. Then, the input is a temperature weather forecast and the output is also a temperature weather forecast. However, the output is produced according to the MM5 model. In this case, the above mentioned service has in its W^{U} ontology the following relationships:

{**Web Service**, hasEffect, **MM5**},{ **Web Service**, inParameter, **NOGAP**},{ **Web Service**, inParameter, **MM5**},{**Forecast**, isInputTo, **Web Service** },{ **Forecast**, isOutputTo, **Web Service** }

Thus, when the semantic viewer cannot match the userCriteria to an instance in the augmented domain ontology, it queries the augmented domain ontology for a Web service whose Effect matches the user criteria. If the service is found, the semantic viewer invokes it to produce the desired instance data, and then proceeds with its processing as in 2.4.

3 Automatic Code Generation

The Semantic Viewer identifies executable paths through the graph to satisfy a given request. Once an SME chooses one, we would like the system to automatically

generate code that results in the request being satisfied. Technology exists to generate code from the executable path. This Code Generator is created once for $\mathbf{D}^{++}$ and is used many times.

To implement the Code Generator, we used the CodeDom [15] technology provided by .Net from Microsoft. CodeDom builds on .NET Reflection to generate high level source code (C, Java, or CSharp as is the case here), to instantiate an instance compiler, and to compile the code and link it to other system libraries in real-time.

The produced code is a web service that includes several elements: a JavaScript file containing the code for specifying the inputs and output and for invoking itself; the code necessary to retrieve data from the database; the code that forms the (Mapquest) WS URL with the data properly inserted into its parameters.The net effect is a dynamically generated web service that implements the integration of the database with Web services. This Web service is then deployed and ready to serve clients, and the Web service code is self-contained. i.e. it does not require the services of the OMS from this point on. Consider the scenario of the previous section where a user wishes to map the location of a business but he can only provide the business name. In this case, the semantic viewer displays the graph shown in Figure 10. Then the Web service generator implements a Web Method [16] with the values of userInput and userCriteria as the input arguments, and with return value as the URL of the MapQuest service with all of its parameters filled in with the right values. Then the Web service generator emits the body of this method by walking the graph starting from the Web service and stopping when the code for retrieving the values of input parameters has been emitted. At this point the URL is formed and the Web Method terminates.

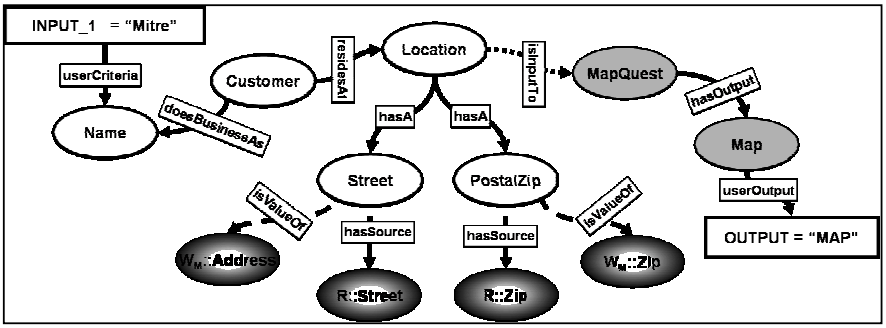

Figure 10 – Graph To Generate a Web Service Code

4 Relation to other work

4.1 Comparison to OWL-S

The approach that we have developed differs from the OWL-S approach in an important way: OWL-S does not address web service integration with an existing database without first building a web service that abstracts that database. In our

approach, a web service can be integrated with a database (or any other software component for that matter) through simply linking ontologies. Thus there is no preferential integration methodology – all software components are on an equal footing through their ontological representation.

The Web service ontologies that we have developed are similar to the OWL-S web service ontology model. The upper domain ontology for a Web service contains information similar to the ServiceProfile of the OWL-S web service ontology model.

In the OWL-S ServiceProfile the conditions are logical conditions that must be satisfied prior to the service being requested. And the effects are the result of the successful execution of the service. The required conditions and the expected effects are interpreted within the OWL-S ServiceProfile as specifying the state change produced by the execution of the service. For the $\mathbf{W}^{\mathbf{U}}$ ontology the use of conditions and effects are somewhat different from the OWL-S approach. The conditions specified by $\mathbf{W}^{\mathbf{U}}$ relate inputs of Web services to the concepts in the domain ontology. These conditions must be satisfied in order that the Web service be able to operate on an instance of the domain D++. The Effects as specified in $\mathbf{W}^{\mathbf{U}}$ reflect property changes on an instance that occur as a result of running the Web service on a particular instance data in the domain ontology.

The upper ontology $\mathbf{V}^{\mathbf{U}}$ information is similar to the ServiceGrounding of the OWL-S web service ontology model The OWL-S ServiceModel provides abstract descriptions of the properties, inputs and outputs etc., of the processes of the Web service. The OWL-S ServiceGrounding provides a mapping of the abstract descriptions contained within the ServiceModel to the concrete messages that carry the inputs and outputs in a transmittable format. This mapping is done in an XSLT. In our approach, the mapping is done using links or OWL ObjectProperty. The $\mathbf{V}^{\mathbf{U}}$ ontology described here is limited to atomic processes and does not provide the capability for describing composite processes.. However, the combination of processes is still possible. That is, it is the Semantic Viewer that infers which processes can be combined in order to satisfy a request and suggests a set of executable paths to the SME. Then, the SME selects one such path and persists the code that implements it. We also don't use control constructs as specified by OWL-S.

Another aspect of this methodology is our ability to link a user's input to various concepts in the domain ontology using userCriteria and userInput. These two relationships provide a mean to interpret a user's input. To illustrate this, consider the following:

> A user's INPUT_1= "01851", and OUTPUT= "Map", and the SME links INPUT_1 to zip-code using the userInput relationship. In our methodology and in OWL-S, this would be interpreted as "map the area having zip-code of 01851". However, if the SME links INPUT_1 to zip-code using userCriteria, then our system returns the maps of all occurrences of addresses (within the database) where zip-code = 01851.

4.2 Comparison with Middleware Solutions

Traditional middleware technologies generate proxy/stub [17] code that hides distributed complexities (e.g., method invocation, marshalling, etc), but they do not generate glue code to integrate software components. Unicorn Solutions, using

Semantic-based middleware technology, translates between different XML schemas using a unifying information model [18]. The information model is active in that it can generate the transformation code from any one schema to any other, thus producing the XSL style sheet or the SQL but still doesn't integrate software components. By contrast, our solution does.

5 Summary

This work demonstrates the utility of building domain ontologies for information systems. By representing software components ontologically (to characterize the structure of data), and by linking these component ontologies to the domain ontology, we effectively integrate the software components. The execution paths that result from extracting a sub-graph from the ontologies not only represent glue code, but it also specifies the semantic agreement that must be achieved before an integration can take place. Our future work will articulate how semantic agreement can be achieved as a result of adopting this methodology. Today, the semantic agreement takes years to accomplish for disparate information systems. Having used this framework to implement two prototypes we can offer the following insights regarding the utility of OWL and RDF. This approach relies heavily on the existence of named relationship in the RDF triple. This is evident in the linking between the ontologies. Without this formalism, it is virtually impossible to make this approach work. We also make heavy use of two inferences: The "IS-A", and "Inverse-of" inferences which are fundamental to the working of the compose service. Also, the ability to combine both instance data (RDF) and schema (RDF S.) proved very useful in prototyping the Semantic Viewer and the Web service generator. We have established the methodolgy for using semantic web technologies to integrate software components. Next we will extend this approach to work with multiple domain ontologies in which each is mapped to multiple software components.

6 Acknowledgement

This work was sponsored under AF Contract **FA-8721-04-C0001.** The authors would like to acknowledge Scott R. Bennett for his contribution to the comparisons to OWL-S, and to thank Scott and Vuhuy N. Phan for their able coding of the Semantic Viewer and their handling of the OMS software.

References:

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web, Scientific American, May 2001. T. Berners-Lee, Hendler J., and O. Lassila.

[2] Michael C. Daconta, Leo J. Obrst, and Kevin T. Smith. The Semantic Web : A Guide to the Future of XML, Web Services, and Knowledge Management

[3] World Wide Web Consortium. http://www.w3.org

[4] T.R. Gruber. A Translation Approach to Portable Ontologies. J on *Knowledge Acquisition, Vol 5(2), p199-220, (1993)*

[5] Guarino, N.; Giaretta, P. 1995. Ontologies and Knowledge Bases: Towards a Terminological Clarification. In: N. Mars, ed. Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing. IOS Press, Amsterdam: 25-32
[6] Web Ontology Working Group. http://www.w3.org/2001/sw/WebOnt/
[7] Resource Description Framework (RDF). http://www.w3.org/RDF/
[8] C.J.Date, An Introduction to Datsabase Systems, Eight Edition, Addison Wesley, Boston, MA, 2003.
[9] http://www.w3.org/2002/ws/desc/
[10] Werner Ceusters, Peter Martens, Christoffel Dhaen and Boris Terzic. Linkfactory: an advanced formal ontology management system. Victoria, Canada October 2001. K-CAP 2001
[11] Language and Computing. http://www.landc.be
[12] Mapquest, http://www.mapquest.com
[13] OWL-S Technical Overview. http://www.daml.org/services/daml-s/0.7/
[14] Sabbouh M., Pulvermacher M, Wedding the Web: An Example of a Services and Semantics Marriage that Works. To Appear in SCI 2004, Orlando FL.
[15] .NET Framework Developer's Guide, - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconUsingCodeDOM.asp
[16] Paula Paul, Web Methods Make it Easy to Publish Your App's Interface over the Internet,http://msdn.microsoft.com/msdnmag/issues/02/03/WebMethods
[17] Shapiro., Structure and Encapsulation in Distributed Systems: the Proxy Principle". ICDCS, Cambridge MA (USA), May 1986
[18] Joshua Fox. Active Information Models for Data Transformation. EAI journal May 2003
[19] Roy Fielding, Ph.D. Dissertation, http://www.ebuilt.com/fielding/pubs/dissertation/top.htm
[20] World Wide Web Consortium, XML Protocol Working Group, http://www.w3.org/2000/xp/Group/