HARMONY - A FRAMEWORK FOR AUTOMATIC WEB SERVICE COMPOSITION

Viorica R. Chifu, Ioan Salomie
Department of Computer Science, Technical University of Cluj-Napoca, Romania

Emil Şt. Chifu, Constantin Pârţac
Department of Computer Science, Technical University of Cluj-Napoca, Romania

Keywords: Web service, Web service composition, semantic Web, ontology.

Abstract: Web services are software components that were designed to improve interoperability and integration of applications developed on different platforms. Web Service composition offers the facility to create new services out of the existing services satisfying a complex functionality. This paper presents HARMONY, a framework for automatic Web service composition. Our approach for automatic Web service composition is based on the GraphPlan algorithm. In HARMONY we use ontologies for the semantic annotation of Web services, so that the automatic service discovery, composition and execution can be realized based on ontology inference.

1 INTRODUCTION

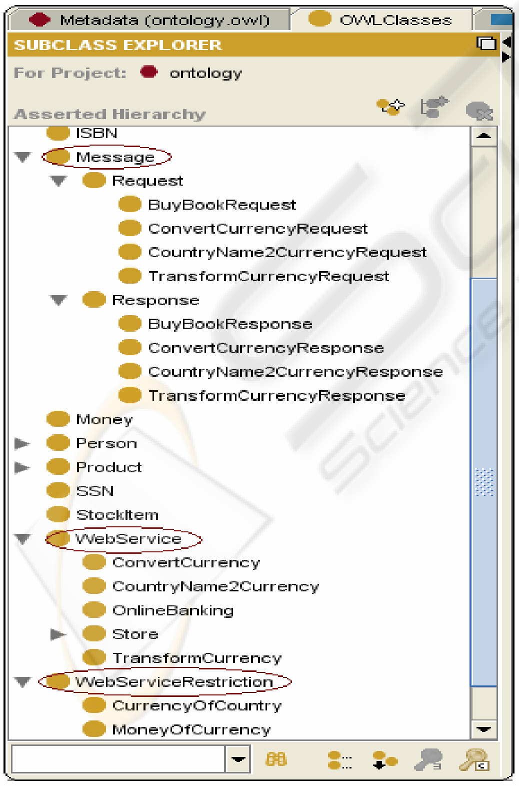
Web services provide a standard way to ensure the interoperability among different software applications running on a variety of platforms. The current standard technologies for Web services provide descriptions only at the syntactic level of their functionality, without any formal description of their semantics. This drawback prevents the use of Web services in complex business contexts, where the automation of these business processes is necessary. Semantic Web Services (Akkiraju, 2005) (Lausen, 2005) enhance WS standards by annotating services with semantic descriptions provided by *ontologies*.

This paper presents HARMONY, a framework for automatic Web service composition and execution based on ontologies. Our approach for automatic Web service composition is based on the GraphPlan (Blum and Furst, 1995) algorithm. The paper is organized as follows. Section 2 presents the ontology model. The framework architecture and implementation is briefly described in section 3, Conclusions and future directions are presented in section 4.

2 ONTOLOGY

Our ontology model contains *classes*, *individuals and properties*. The classes are concrete representations of domain concepts. A property either defines a relation between concepts or a restriction. There are two types of relations in our ontology model: hierarchical relations and non-hierarchical relations. The hierarchical relations are taxonomic relations while the non-hierarchical relations are relating concepts across the hierarchical structure. Restrictions describe a class of individuals and possibly a number of relationships that they participate in. In our ontology model we have defined existential restrictions. An existential restriction describes the class of individuals that are in relationship with at least one individual member of another class.

Three main generic classes can be identified as the core of our model: *WebService*, *Message* and *WebServiceRestriction*. WebService class is the root class of the service classification tree. The *Message* tree has two generic classes of concepts: *Request* and *Response*, which are classifications of the inputs and outputs of the services respectively. Finally, the *WebServiceRestriction* tree is a classification of the effects and preconditions of the services. The generic properties of the Web services which are taken into account by our model are the following: (i) the **endpoint** as a data type property indicating the address at which the service can be invoked; (ii) the **input** as an object property representing a request message as an output from another service; (iii) the **output** as an object property representing a response message; (iv) **precondition** and **effect** as object properties representing conditions on the information space before and after the services are executed; (v) **description** as a data type property representing the description of a Web service. Based on this ontology model we have developed an ontology for an *online bookstore* using the Protégé OWL editor (Horridge, 2004) (Figure 1). This ontology is used for semantic annotation of web services, as a vocabulary for the graphical user interface and for a "smart" planner. In our work, we use SAWSDL (Verma, 2007) for the semantic annotation of Web services. Because at this moment there isn't any final decision about representing the preconditions and effects in SAWSDL, we have chosen to extend SAWSDL with WSDL-S (Akkiraju, 2005) schema, which provides a way of representing the preconditions and effects.


Metadata (ontology.owl) | OWLClasses

SUBCLASS EXPLORER

For Project: ontology

Asserted Hierarchy

```
ISBN
▼ Message
    ▼ Request
        BuyBookRequest
        ConvertCurrencyRequest
        CountryName2CurrencyRequest
        TransformCurrencyRequest
    ▼ Response
        BuyBookResponse
        ConvertCurrencyResponse
        CountryName2CurrencyResponse
        TransformCurrencyResponse
Money
► Person
► Product
SSN
StockItem
▼ WebService
    ConvertCurrency
    CountryName2Currency
    OnlineBanking
    ► Store
    TransformCurrency
▼ WebServiceRestriction
    CurrencyOfCountry
    MoneyOfCurrency
```

Figure 1: Main OWL ontology classes.

3 FRAMEWORK ARCHITECTURE

HARMONY is an experimental framework for automatic Web service composition created to ease the process of composition, thereby reducing the complexity and the development time of a composite Web service. HARMONY components were designed to be independent of each other. An overview of the HARMONY architecture is presented in Figure 4, where the arrow connections represent the data flow.

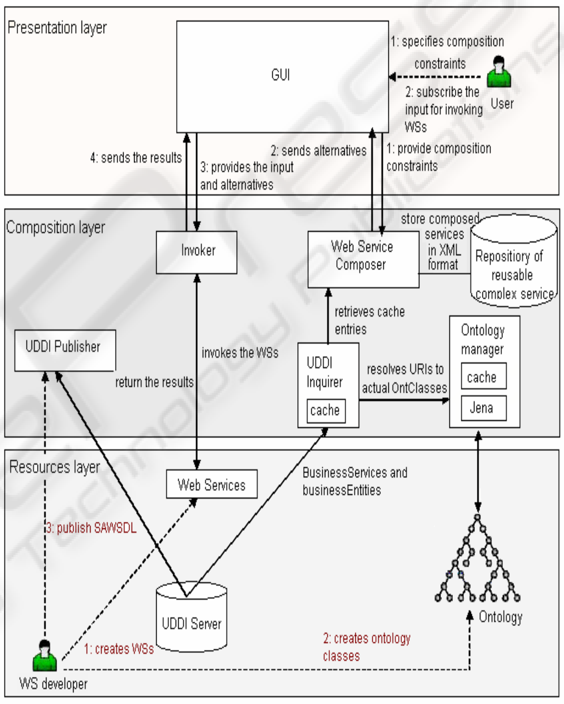

Presentation layer
GUI
1: specifies composition constraints
2: subscribe the input for invoking WSs
User
4: sends the results
3: provides the input and alternatives
2: sends alternatives
1: provide composition constraints
Composition layer
Invoker
Web Service Composer
store composed services in XML format
Repository of reusable complex service
retrieves cache entries
UDDI Publisher
invokes the WSs
return the results
UDDI Inquirer
cache
resolves URIs to actual OntClasses
Ontology manager
cache
Jena
Resources layer
Web Services
BusinessServices and businessEntities
3: publish SAWSDL
UDDI Server
1: creates WSs
2: creates ontology classes
Ontology
WS developer

Figure 2: Framework architecture.

The *UDDI Publisher* takes the WSDL as input and creates its corresponding tModel, bindingTemplates and businessServices into the UDDI server. As *UDDI server* we use jUDDI provided by the Apache Foundation (jUDDI, 2007). The *UDDI Inquirer* creates a cache of all the registry entries. This way, inside the *UDDI Inquirer cache*, we create the mappings of the inputs, outputs, preconditions and effects of Web services to ontology classes. The *Ontology Manger* consists of the Jena API framework (Carroll, 2004) and a cache used to store the ontology model. The GUI interacts with the *Web Service Composer* and *Invoker* modules. The user only needs to select by the GUI the inputs, outputs, preconditions and effects (which are concept classes from the ontology) of the desired services and to invoke *Web Service Composer*. After the planner finds the solution, it should be evaluated and validated, and then *Web Service Invoker* is called in order to execute the newly composed Web Service. A more detailed description of each component is presented in what follows.

3.1 Ontology Manager

The *Ontology Manger* uses the Jena Ontology API (Carroll, 2004). The main drawback of Jena resides in its speed inefficiency. In order to improve the performance, our *Ontology Manger* has a caching mechanism which stores the ontology model between Jena interface calls. It allows for a significant speed improvement mainly because the same model is used by all of the framework modules interacting with the ontology. If a new rule needs to be inferred, the average loading time of the ontology is between 2 and 5 seconds. By using the Jena caching mechanism, a retrieval of the ontology model takes approximatively 250 ms, but by using our Ontology Manager caching mechanism it takes up to 3 ms.

3.2 UDDI Publisher

A simple method for publishing Web services was implemented in order to add a high degree of automation to the publishing process. In our approach we publish a service by simply providing the SAWSDL files to our *UDDI Publisher*. The *UDDI Publisher* consists of two modules: *Interface Publisher* and *Service Publisher*. *Interface Publisher* is responsible for publishing Service interfaces which are mapped into UDDI tModels. The *Service Publisher* is responsible for publishing the Service implementation which is mapped into businessServices and bindingTemplates.

3.3 UDDI Inquirer

There are remarkable implementation efforts in the area of service composition frameworks in general and in UDDI query in particular. The main disadvantage of previous approaches (Châtel, 2006 and Verma, 2006) is that they are constrained to using UDDI defined inquiry facilities. We took an alternative approach by defining a separate UDDI inquirer tool. A cache entry is defined for each bindingTemplate, which corresponds to a Web Service operation. In order to generate the cache entry, the SAWSDL corresponding to a bindingTemplate is parsed and the URIs of the ontology classes representing the input, output, preconditions and effects are extracted. Then, a call to Ontology Manager is made in order to include the actual ontology classes in the cache.

3.4 Web Service Composer

Our Web Service Composer implements the GraphPlan algorithm, which takes into account the inputs, outputs, preconditions and effects. The GraphPlan algorithm takes into consideration only the viable solutions. The GraphPlan algorithm finds such viable solutions by evaluating at each state whether the preconditions necessary to run a service are met and whether all the inputs are present. The java class implementing the GraphPlan algorithm finds all the goals situated on the same depth in the graph, but it can be configured to find the solution until a certain depth is achieved or until it finds a certain number of solutions. In order to be able to reuse the solutions, they must be saved in an easily serializable / deserializable format. The framework saves the solutions provided by GraphPlanner in XML format in a repository of reusable complex services.

3.5 Invoker

The composite service invoker takes as input the list of Web services to be invoked and the values to be passed as inputs and generates an Axis engine client for each call. The invoker parses the SAWSDL files in order to configure the clients and assigns a serializer for each input and a deserializer for each output. Then, it constructs the java bean classes corresponding to each input of the first service. The result of the first service is automatically converted by the Axis engine to a java bean. Then, the second service is invoked, and so on, until the last one. In this execution scenario, a java bean must be generated for each concept class described in the ontology.

3.6 GUI

The graphical user interface is composed out of two functional parts: the composer GUI and the invoker GUI. In the former, the user selects the ontology whose concepts he wants to use in the automatic composition of Web Services, followed by selecting the inputs, outputs, preconditions and effects of the desired service. The inputs, outputs, preconditions and effects are chosen from the previously selected ontology classes. After invoking the service composer, the invocation interface is presented, where the available solutions are shown, allowing the user to select the input values for the composed service.

4 CONCLUSIONS

In this paper we have presented HARMONY, a framework for automatic Web service composition based on ontologies. In our approach, the ontology is used in all the steps of the automatic composition process: for the semantic annotation of Web services, as a vocabulary for the graphical user interface, and for implementing a "smart" planner by using semantics. The proposed solution is rather generic, and could be used in different contexts. Framework components were designed to be independent of each other, so that we can add / replace framework components without major modifications. The user interface is simple, but at the same time powerful and intuitive. It drives the user in the composition process in a friendly manner by providing a controlled language that uses the ontology concepts. The only task for the user when he desires a new composed Web service is to change the specification, i.e. the input, output, preconditions and effects. In future work, we plan to extend our framework with a QoS module in order to allow for dynamic selection of the best solution from the set of solutions generated by the planner. Another important effort will be directed towards the addition of heterogeneity handling. This would allow using semantically compatible concepts from different ontologies.

ACKNOWLEDGEMENTS

This work was supported by the PI-1020 (333/2007) project within the framework of the "Research of Excellence" program initiated by the Romanian Ministry of Education and Research.

REFERENCES

Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M.-T., Sheth, A., Verma, K., 2005, http://www.w3.org/Submission/WSDL-S/.

Bechhofer, S., van Harmelen, F., et al., 2004. OWL web ontology language reference. W3C recommendation, M. Dean, G. Schreiber (eds.).

Blum, A. and Furst, M., 1995. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pp. 1636-1642.

Cardoso, J., Sheth, A., 2003. Semantic e-Workflow Composition. *Journal of Intelligent Inform. System*, 21(3): pp. 191-225.

Carroll, J., Dickinson, I., Dollin, D., Reynolds, D., Seaborne, A., Wilkinson, A., 2004. Jena: Implementing the Semantic Web Recommendations. *Proceedings of the 13th World Wide Web conference on Alternate track papers & posters*, New York, NY, USA, pp. 74-83

Châtel, P., 2006. WSDL 2.0 to UDDI mapping SAWSDL to UDDI mapping, Thales Group.

Farshad, H., et al., 2005. Semantic Web Service Composition in IRS-III: The Structured Approach, in *Proc. of the 7 IEEE Intl Conf on Ecommerce Technology (CEC'05)*.

Haarslev, V., Möller, R., 2003. Racer: An OWL reasoning agent for the Semantic Web. *Proceedings of the International Workshop on Applications, Products and Services of Web-based Support Systems*, pp: 91-95.

Lausen, H., Polleres, A., Roman, D., (Eds), 2005. Web Service Modeling Ontology (WSMO). W3C Member Submission, http://www.w3.org/Submission/WSMO/.

McIlraith, S. and Son, T., 2002. Adapting Golog for Composition of Semantic Web Services. *In the Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning* (KR2002).

Roman, D., et. al. , 2005. Web Service Modeling Ontology. Applied Ontology 1 (2005) pp.77–106

Sirin, E., et al., 2005. Template-based Composition of Semantic Web Services, *AAAI Fall Symposium on Agents and the Semantic Web*, Virginia.

Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., and Katz Y.,2006. Pellet: A Practical OWL-DL reasoner. *In the Journal of Web Semantics*.

Verma, K., Gomadam, K., Sheth, A. P., Miller A.J., Wu, Z., 2005. The METEOR-S Approach for Configuring and Executing Dynamic Web Processes, Technical Report.

Verma, K., 2006. Configuration and Adaptation of Semantic Web Processes. PhD thesis, Department of Computer Science, University of Georgia.

Verma, K., Sheth, A. P, 2007. Semantically Annotating a Web Service, *IEEE Internet Computing*, pp. 83-85B.

Horridge, M., Knublauch, H., et al., 2004. A practical guide to building OWL ontologies using the Protege-OWL plugin and CO-ODE Tools. The University Of Manchester.

Wu, Z., Ranabahu, A., Gomadam, K., Sheth, A.P., Miller, J.A.,2007. Automatic Composition of Semantic Web Services using Process and Data Mediation, LSDIS lab, University of Georgia.

jUDDI, http://ws.apache.org/juddi/, accessed June 2007