Personalized Service Discovery and Composition

Hua Xiao
School of Computing
Queen's University
Kingston, Ontario, Canada
huaxiao@cs.queensu.ca

Ying Zou
Dept. of Electrical and Computer Engineering
Queen's University
Kingston, Ontario, Canada
ying.zou@queensu.ca

Joanna Ng, Leho Nigul
Center for Advanced Studies
IBM Toronto Lab
Markham, Ontario, Canada
{jwng, lnigul}@ca.ibm.com

Abstract

With the prevalence of Service-Oriented Architecture (SOA), end-users, who are not familiar with SOA standards and tools, want to customize services to fulfill their daily activities. However, it is challenging for end-users to get involved in the service composition, due to the in-depth knowledge needed for SOA standards and techniques. To help end-users compose services for their daily activities, we propose an approach that shields end-users from the complexity of SOA standards and automatically generates composed services by specifying a goal using keywords. We expand the meaning of a user's goal using ontologies and derive a group of keywords for discovering services in order to achieve a user's goal. We also detect contextual environment of a user and personalize the service composition to meet a user's situational needs. A prototype is developed as a proof of concept to show that our approach enables non-expert end-users to discover and compose services easily.

1 Introduction

In Service-Oriented Architecture (SOA), applications can be assembled by a set of existing distributed services spanning organizations and platforms. SOA increases reuse and cooperation among services. Therefore, applications can be built in a rapid way with relatively low cost. With the prevalence of SOA, end-users, who are not familiar with SOA standards and tools, want to compose and customize services to fulfill their daily activities. For example, planning a trip is a typical goal for many end-users. It involves several tasks, such as searching for flight tickets, booking a hotel, and checking the weather reports for the destination. To achieve such a goal, an ad-hoc process which records the tasks frequently performed by an end-user is formed. In an ad-hoc process, the tasks can be executed without strict order. The execution of an ad-hoc process involves the dynamic integration of various services (e.g., Web services, and Web sites). A task can be associated with more than one service of the similar functionally. For example, the task, "searching for flight tickets", can be implemented by different travel agent services. In the current practices, end-users manually search and visit several Web services to gather each piece of information for planning a trip. They potentially compose an ad-hoc process to fulfill their needs. However, it is challenging for end-users to get involved in the service composition, due to the in-depth knowledge needed for SOA standards and techniques.

To shield end-users from the complexity of Web services standards and tools, we propose a system architecture that automatically generates an ad-hoc process for a user and customizes the process to reflect the user's situational needs by detecting the user's contextual environment. More specifically, context refers to the information that characterizes the situation of a person, place or the interactions between humans, applications and the environment [4]. We define a context model to capture the contextual data of users. Instead of requiring end-users to specify the concrete tasks, our approach only need end-users to describe the goal using keywords. We expand the meaning of a user's goal using ontologies and derive a group of keywords for discovering services in order to achieve a user's goal. Once we detect contextual data, we infer rules from ontologies that are formed from the contextual data. The rules describe relations between the contextual data and the possible actions to be performed by a user. We use the rules and suggested actions from the rules to personalize service discovery and recommendation. The recommended tasks refines the ad-hoc process and provide solutions for individual users' daily activities.

The remainder of this paper is organized as follows. Section 2 introduces the background of ontologies. Section 3 discusses our proposed architecture. We describe the major components in the architecture. Section 4 describes a proof of concept prototype that implements the proposed architecture. Finally, Section 5 concludes the paper and presents the future work.

2 Modeling an Ontology Definition

An ontology expresses common concepts (e.g., people, travel and weather), and the relations among those concepts. Figure 1 illustrates an example ontology for defining the concept "travel". The concept "travel" is related to three sub-concepts: "Transportation", "Accommodation", and "Weather Forecast".

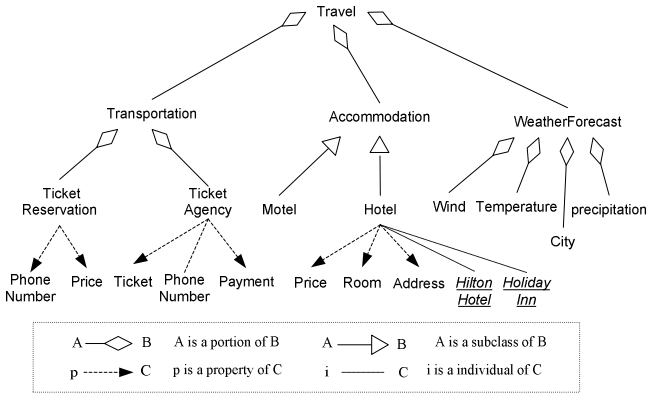

Figure 1. An Example Ontology

Ontologies can be described using different ontology specification languages, such as Web Ontology Language (OWL) [19] and Resource Definition Framework (RDF) [5]. We create a generic ontology model to capture the structures and concepts common to various ontology specification languages. The major components in our generic ontology model are listed as follows:

- *Class*: is an abstract description of a group of resources with similar characteristics. For example, "Hotel" is a *class* which depicts the common characteristics of different hotels. *Class* is also called "concept", "type", "category" and "kind" in various ontology languages.
- *Individual*: is an instance of a *class*. For example, "Hilton Hotel" is an instance of *class* "Hotel", i.e., "Hilton Hotel" is an *individual*.
- *Relation*: defines ways in which *classes* and *individuals* can be related to one another. Typical *relations* include *subclass*, *partOf*, *complement*, *intersection* and *equivalent*. *Subclass* describes that a class extends an abstract class to convey more concrete knowledge. *PartOf* relation indicates that a class/individual is a part of another class/individual. *Complement* selects all members (classes or individuals) from the domain of the ontology that do not belong to a certain class. *Intersection* describes a class which contains precisely the members described in all classes in the class list. *Equivalent* expresses that two classes contains exactly the same set of individuals.
- *Attribute*: describes a property of a *class* (and an *individual*). *Attribute* is also called "aspect", "property", "feature" or "characteristic" in various ontology specification langauges. For example, "Price" is an *attribute* of *class* "Hotel".

3 An Overview of System Architecture

Figure 2 provides an overview of our proposed system architecture for generating and personalizing an ad-hoc process. The architecture is built using client/server architecture. In the client side, a service composer panel provides a user interface to enable end-users to specify the goal, navigate through the generated ad-hoc process, edit the process, and select services. Context sensors monitor the user's activities and capture user's contexts. In our work, the context sensors are deployed as a plug-in into various applications, such as web browsers and on-line calendars running on the user's computing environment.

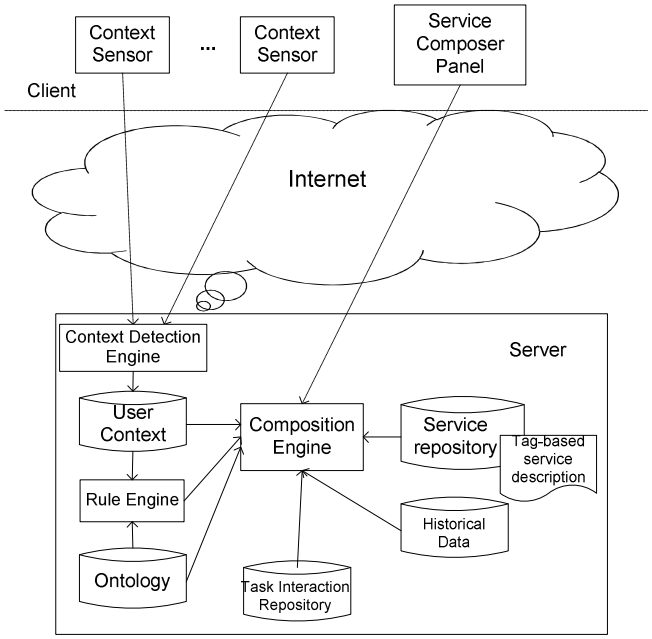

Figure 2. An overview of System Architecture

The server contains several components that process the contextual information gathered from the client and generate an ad-hoc process by interacting with the service composition panels. The components are described as follows:

Service repository allows service providers to advertise their services and provide interfaces for automatic service discovery. To enable end-users and composition tools to understand the properties of Web services, we propose a service description schema that describes services using descriptive tags (i.e., keywords). The detailed information of the tag-based service description schema is described in our earlier publication [10].

Composition engine receives the request from end-users and automatically composes a personalized ad-hoc process. Specifically, a user simply describes a desired goal using keywords. The composition engine uses the goal description to find a matching ontology. In an ontology, the semantic of a high-level goal is expanded into more concrete concepts. The composition engine uses the concepts as keywords to search for services in a service repository. To facilitate the reuse of the services when the same goal reoccurs, we abstract the discovered services into tasks and aggregate tasks into an ad-hoc process. The ad-hoc process can be stored and shared among multiple end-users. The algorithm for generating ad-hoc processes is published in our earlier work [8].

Task interaction repository records the execution orders of tasks while end-users perform an ad-hoc process. We track the execution order of tasks based on a user's selection and store such information from different users in a repository. To guide a user to perform the tasks, we conduct statistical analysis on the historical data to detect the control flows (e.g., alternative, sequential and iterative relations) among tasks.

Context detection engine communicates with various context sensors on the client side to receive the updates of contextual data, such as user preferences, local time, and the agenda of users.

Rule engine is used to conduct the contextual analysis to recommend services. The rule engine searches for ontologies for each piece of contextual data. By examining the relationships among different ontologies of contextual data, the rule engine can refine the result of service discovery and recommend personalized tasks to the users. The identified personalized tasks are automatically added to the ad-hoc user process.

In the following sub-sections, we discuss our context model and rule engine in more detail.

3.1 Context Modeling and Detection

We define a context model to depict the contextual data. There exist many context models [7][22] in literature. However, most context models are proprietary to a particular system. It lacks of standard ways to represent context models. An existing context model needs to be redefined by changing parameters and structures in order to adapt it to a new environment. The mappings between different context models need to be created in order to exchange contextual data between different systems.

To create a flexible context model for representing the contexts of different users, we describe the context model using the proposed generic ontology model. A contextual data (e.g., the location of the user) is denoted as a concept in the generic ontology model. The relations between different contextual data can be described in the same way as the relations among concepts in the generic ontology model. To modify the details of a context model, we can extend an existing branch of a concept by importing the new context definition.

Figure 3 depicts an example context model specified using our generic ontology model. The context model describes the contextual data for each user, such as user profile, agenda, current time and computing environment. Computing environment describes the hardware platform (e.g., mobile device), and the network condition. User profile maintains the user's information. A user's goal is also included as part of the user's context. In the example of "Travel to New York", the goal could include the location context "New York" and the activity "Travel".

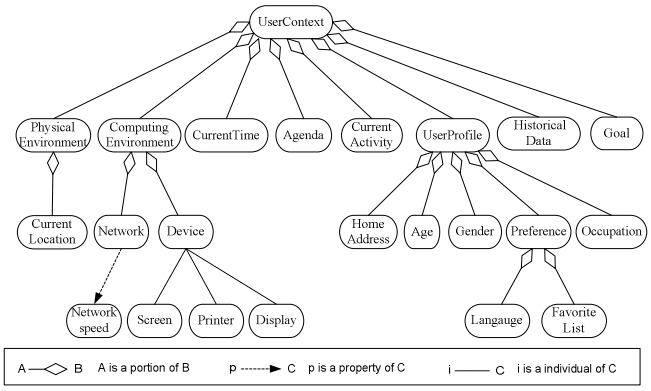

Figure 3. A Context Model

We design and develop a set of context sensors to detect contextual data defined in the context model. When a new is added into the context model, we need to provide the relevant sensor to detect the new contextual data. In the current stage of our work, we detect contextual data from the following software components in the user's computing environment:

(1) ***Operating system:*** We extract information, such as time zone, IP address, regions and language options, from the system configuration of the operating system, and use them as the default values for the contextual data relevant to the user's physical environment, such as Current Time, Current Location and Network Speed.

(2) ***Applications:*** Applications, such as Microsoft Outlook and Google calendar, are generally used to assist users to manage and accomplish users' daily activities through emails, task list and agenda definition. The data in such applications help us infer the details of users' goal. For example, the travel plan can be marked in the user's calendar.

(3) ***Web browser:*** the bookmarks and historical access information logged in the cache of a web browser reflect the user's preferences and reoccurring on-line activities.

(4) ***Specific devices:*** devices, such as GPS can provide more accurate contextual data.

3.2 Rule Inference from Ontology

Rules are generally pre-defined to connect the contextual scenarios to the possible user's behaviors. However, fixed rules are not flexible enough to accommodate the changing environment and various personal interests. To provide a general approach for personalizing service discovery and composition, we propose an approach that dynamically generates rules using contextual data. To infer rules, we identify the relations between user contextual data in the following steps:

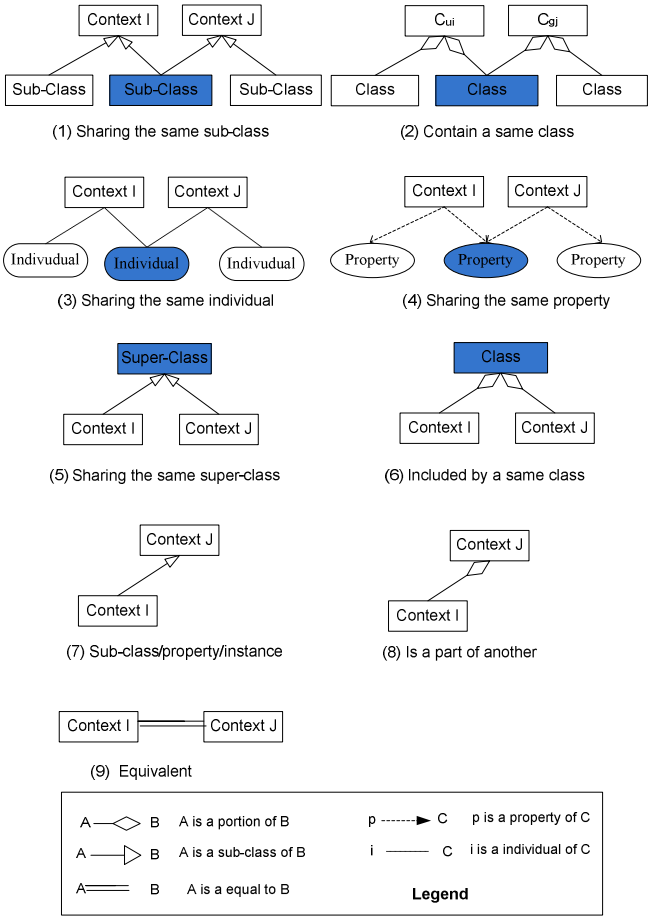

Figure 4. Relations among Contextual Data

1) Indentify the ontology related to contextual data.
In the context model, each piece of contextual data is described as pair *(Context_Type, Context_Value)*. For example, the contextual data "home address" could include a pair *(City, Toronto)* to depict the city of an address. We extract keywords from context type and context value, and use these keywords to search for relevant ontologies in the ontology repository. Since we treat goal itself as a part of contextual data, we reuse the ontology which we use to generate ad-hoc process from the goal.

2) Group and merge ontologies of context data.
In the context model, a goal represents the information of current composed ad-hoc process. To use contextual data to recommend new tasks which are relevant to the ad-hoc process, we need to find the relations between goal and other contextual data. We split the ontologies of the contextual data into two groups:

(I) The ontology of the goal, i.e., *Set(Cg) = {Ontology of Cgi | Cgi is a piece of contextual data belonging to a goal.}*. For example, if the goal is "Travel to New York", it includes the ontology of "Travel", and ontology of "New York".

(II) The ontologies of user contextual data (except the goal), i.e., *Set(Cu) = {Ontology of Cui | Cui is a piece of contextual data, but does not belong to the goal. }*. Examples of *Set(Cu)* are the ontologies for user's preferences and current location.

3) *Discover the relationships between user contextual data and the goal.*
We detect the relationships between the user context and the user's goal via comparing the ontologies in *Set(Cg)* and *Set(Cu)*. We identify 9 patterns that describe the possible relations among contextual data shown in figure 4. Generally, these 9 patterns of relations are classified into four categories:

(I) *Sharing common entities in the ontologiess*: if two ontologies of two contextual data contains the same set of entities such as sub-classes, individuals, properties, or classes, we say that these two contextual data share the common ontology entities. As shown in figure 4(1), figure 4(2), figure 4(3), and figure 4(4),

(II) *Sibling*: We call two contextual data are *sibling*, if they have the same super-class or both of them are a part of the same class. Contextual data in Figure 4(5) and figure 4(6) are in the *sibling* relation.

(III) *Subsume*: If one contextual data is a sub-class, a property, an individual or a part of another contextual data, we say these two contextual data have a *subsume* relation. Figure 4(7) and figure 4(8) represent the *subsume* relation.

(IV) *Equivalent*: two contextual data have exactly the same definition in ontologies, as shown in in Figure 4(9).

We infer rules in two different levels of abstractions: general rules that can be applied to the users with the same context types; and specific rules that are relevant only to the user with specific context values. We analyze the ontologies of the two contextual data to recognize the relation patterns among the two contextual types. The context value is used to parameterize the general rules. For example, two context types are Fight and Destination City; and their corresponding values are Air Canada and Toronto. We can find that

they share the same concept "Airport". Therefore, we can define the following generic rule to recommend a new task that searches for the public transportation in Toronto airport.

Rule: ***if*** *(Destination = = Toronto*
&& Fight == Air Canada)
Then
search for public transportation in Toronto Airport in Toronto;

The inferred general rules can capture the domain specific service recommendation. However, the generic rules cannot capture all the characteristics of individual user's contexts. For example, a user plans to "Travel to New York" and we know the user likes the "NBA (the National Basketball Association)", it is difficult to guess that there is a NBA team in New York using the generic rules. Therefore, we design the specific rules to capture the detailed relations among context values. The system can automatically recommend new tasks following the same steps as inferring the general rules. In the aforementioned example, the ontologies of "New York" and "NBA" would tell us that they share the same concept "New York Knicks" (i.e., an NBA team). So we can automatically generate a specific rule for the user and recommend a new service relevant to "New York Knicks". Because the user is a NBA fan, he/she should be interested in the information of "New York Knicks" term when he/she travels to New York.

4 Implementation

To evaluate the feasibility of our proposed approach, we built a prototype to help end-users to generate ad-hoc processes. We developed a Firefox extension to capture the contextual data when a user uses the browser to surf the web as shown in figure 5. This Firefox extension can detect the contextual data from Windows operating system, Firefox historical visited websites, Firefox bookmarks and Google online Calendar. We use the IBM WebSphere Service Registry and Repository (WSRR) [13] to register and manage Web services. To display the interfaces of selected services and invoke services, we use the IBM Mashup Center [11] as a service Mashup platform to integrate various Web services.

Figure 6 is an annotated screenshot for service composer panel. A user can specify their goal (e.g., plan a trip to New York) in the *Goal Editor*. In our current implementation of the prototype, the ontologies are manually searched using Swoogle [20] and Freebase [6], search engines for ontologies, and imported into our prototype to ease the analysis. A personalized ad-hoc process is automatically generated to include a set of tasks that meet the specified goal shown in the *Process Editor*. A task in a generated ad-hoc process can be associated with one or more services. As shown in Figure 7, once a user selects the "*Car Rental*" task in the *Process Editor*, the associated services are automatically displayed in the *Service Selection Panel* on the right side of the markup page. We allow a user to select the most desirable services.

A user can refine and customize the ad-hoc process in the process editor. A user can remove a task if it is not needed by selecting the "*Remove*" check box. A user can also add a new task by specifying keywords for searching for services. We record the modifications as the user's preferences. When a user specifies the same goal, our prototype provides the previously refined ad-hoc process.

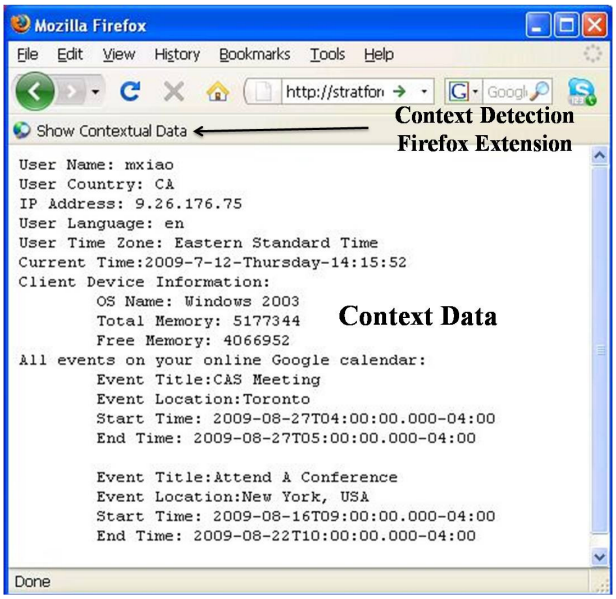

Figure 5. Annotated Screenshot for Context Detection

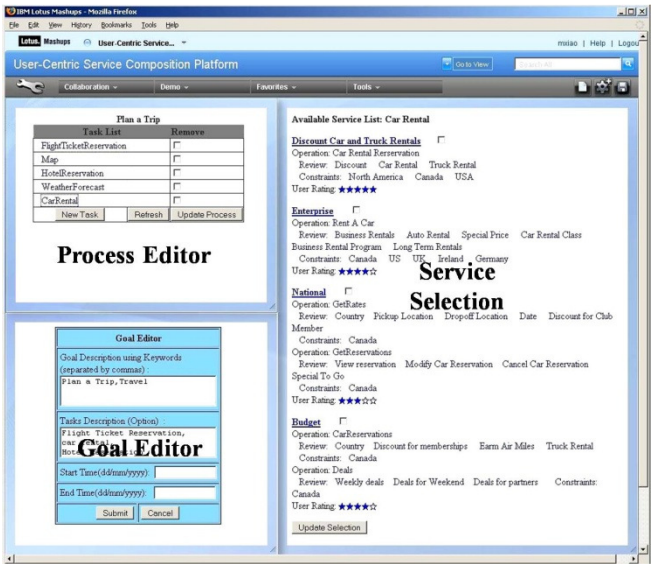

Figure 6. Annotated Screenshot for Service Composer Panel

5 Conclusion and Future Work

In this paper, we present our preliminary system architecture that dynamically generates a user's ad-hoc process and automatically customizes the process to fit it with an individual user. Our approach hides the complexity of SOA standards and tools from end-users and helps an end-user fulfill their daily activities. In the future, we plan to refine our rule generation approach to reflect more complex relations among contextual data.

Acknowledgements

This work is financially supported by the IBM Toronto Centre for Advanced Studies and NSERC.

References

[1] D. Beckett, B. McBride, RDF/XML Syntax Specification (Revised), W3C Recommendation (2004)

[2] S. Brin, L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine," Intl. Conference on World Wide Web, p.107-117 (1998)

[3] R. Chinnici, J. Moreau, A. Ryman, S. Weerawarana, "Web Service Description Language Version 2.0," W3C Recommendation (2007)

[4] A. K. Dey, et al. Toward a Better Understanding of Context and Context-Awareness. Atlanta, GA, USA, Georgia Institute of Technology, 1999

[5] Facebook, http://www.facebook.com/, last accessed on June 18, 2009.

[6] Freebase, http://www.freebase.com/, last visited on October 14, 2009

[7] C. Hesselman, A. Tokmakoff, P. Pawar, S. Iacob, "Discovery and Composition of Services for Context-Aware Systems," Proceedings of 1st European Conference on Smart Sensing and Context 2006 (EUROSSC 2006), Enschede, The Netherlands, 25-27 October, 2006

[8] Hua Xiao, Ying Zou, Ran Tang, Joanna Ng, Leho Nigul, "An Automatic Approach for Ontology-Driven Service Composition", Proc. IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2009) 2009), Taipei, Taiwan, 14-15 December 2009

[9] M. Klusch, "Semantic Web Service Description", In book CASCOM: Intelligent Service Coordination in the Semantic Web, Birkhäuser Basel publishing, page 31-37 (2008)

[10] M. Klusch, Z. Xing, "Semantic Web Service In the Web: A Preliminary Reality Check", First Intl. Joint ISWC Workshop SMR2 2007 on Service Matchmaking and Resource Retrieval in the Semantic Web, Busan, Korea (2007)

[11] IBM Mashup Center, http://www-01.ibm.com/software/info/mashup-center/, last accessed on June 18, 2009.

[12] IBM WebSphere Integration Developer, http://www-01.ibm.com/software/integration/wid, last access on June 18, 2009

[13] IBM WebSphere Service Registry and Repository, http://www-01.ibm.com/software/integration/wsrr/, last access on June 2, 2009

[14] Web Services Business Process Execution Language, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, last accessed on June 2, 2009

[15] U. Küster, M. Stern, B. König-Ries, "A Classification of Issues and Approaches in Service Composition," International Workshop on Engineering Service Compositions (2005)

[16] D. Martin, et al., "OWL-S: Semantic Markup for Web Services," Technical Report, Member Submission, W3C (2004)

[17] Seekda, http://seekda.com/, last accessed on June 16, 2009

[18] M. Sheshagiri, M. desJardins, T. Finin, "A Planner for Composing Services Described in DAML-S," AAMAS Workshop on Web Services and Agent-Based Engineering (2003)

[19] M. K. Smith, C. Welty, McGuinness, D. L.: OWL Web Ontology Language Guide, W3C Recommendation (2004)

[20] Swoogle, http://swoogle.umbc.edu/, last accessed on August 12, 2009

[21] D. Wu, B. Parsia, E. Sirin, J. Hendler, D. Nau, "Automating DAML-S Web Services Composition Using SHOP2," Intl. Semantic Web Conference (2003)

[22] S. J. H. Yang, J. Zhang, I. Y. L. Chen, A JESS-enabled context elicitation system for providing context-aware Web services, Export Systems with Applications, Volume 34, Issue 4 (May 2008), pages 2254-2266