Semantic Web Services for Activity-Based Computing

E. Michael Maximilien, Alex Cozzi, and Thomas P. Moran

IBM Almaden Research Center,
650 Harry Road, San Jose, CA 95120, USA
{maxim, cozzi, tpmoran}@us.ibm.com

Abstract. Semantic Web services promise the addition of semantics annotations to Web services in a manner that enables automatic discovery, usage, and integration of services as part of every day processes. IBM's unified activity management (UAM) implements activity-centric computing concepts by representing human work in terms of activities that relate to each other using semantic information from the various contexts in which the activities are used. In this paper we explore how, using common domain-specific ontologies, we can make use of the semantic annotations added to Web services and our UAM environment, to produce dynamic and richer Web applications widgets and services.

1 Introduction

Human-based activities are best represented as informal loosely structured and semantically rich processes. Even when work activities are well-structured, for instance, using workflow systems, human realization of such workflows typically results in many variations of the different steps, while the same objectives are achieved. This is due to the executing context, which is difficult to predict or capture in workflows. Additionally, the loose realization is also simply due to human behaviors and work patterns which, unless humans are forcefully constrained, are typically loose and malleable [3].

Previous activity-based systems typically organize activities as shared tasks that can be easily modified and arranged to meet work patterns [4]. In addition to distributed task sharing capabilities, IBM's unified-activity management (UAM) [7, 8] computing environment incorporates the loose and malleable characteristics of human activities by representing activities as first-class OWL [5] instances that are interconnected using a semantic network of relationships representing the context and evolution of the activities.

As the majority of knowledge workers' activities involve some form of Web-based application, system, or services, it's easy to see that a UAM-based applications will necessarily use Web resources or be themselves completely Web-based. The addition of semantics to Web resources and Web services [2, 6] enables opportunities for creating semantically rich UAM-based applications and the ability to automate some parts of these applications (and the activities) using software agents. In this paper we investigate the initial use of semantic Web services (SWS) [6] with our UAM environment.

2 Scenario: Use Case

To motivate how our UAM environment can benefit from SWSs we describe a use case scenario based on a simplified activity domain: reading group activities. Our domain involves a set of individual human actors (e.g., knowledge workers, researchers, or students) involved in sharing reading items. The reading items are varied; they are comprise books, book sections, articles, Web pages, and so on; they are contributed by all members of the group, which are assigned to read these items, comment on the contents, prioritize them, rate them, relate them, and make recommendations for new readings that can complement a particular reading item. An implicit goal of such reading group activities is to create new insights from the group's collaboration that otherwise would not be possible had the readings been done individually and separately.

3 Framework

IBM's UAM environment comprises a RDF datastore which keeps OWL instances for all activities, artifacts, actors, and their relationships according to the UAM upper and domain specific ontologies. To expose a services API to UAM that maintains the domain-specific semantics, we created a UAM operator ontology which allows the definition and generation of Web services representing the operations to create, add, modify, and find UAM objects.

The services parameters are typed using the domain-specific activity ontology. We maintain the semantics of the domain by creating OWL-S [9] *Profile* and partial *Process* descriptions for the generated services that are annotated with a domain ontology and a domain-specific activity ontology.

As an example, for our *Reading Group Activity* ontology we expose SWS with operations to *createReadingActity()*, *addBookReadingItem()*, *modifyReadingItem()* passing attributes such as author, description, and so on, according to ontologies for the domains *Reading Document* and *Reading Group Activity.* The generated services connect to an operator API exposed by the UAM environment which allow programmatic access and manipulation of the OWL instances in the datastore.

3.1 Activity Ontology

Figure 1 illustrates our UAM upper ontology. It constitutes the key concepts and relationships of every UAM-based application. This ontology is typically extended by domain-sepecific concepts and relationships that constitute the activities in that domain. The upper ontology defines three main concepts: (1) *uam:Activity* represents an activity—activities have subactivities, have artifacts, and involve actors; (2) *uam:Artifact* represents all non-agent (non-actor) resources—they are the passive resources that are part of activities; and (3) *uam:Actor* represents all active resources involved in an activity—these include human and software agents.

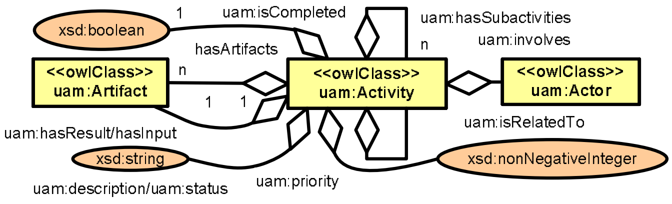


Fig. 1. UAM upper ontology

Activities have other predefined upper-level relationships to represent an activity's description, status, priority, results, and input. Further, every activity can be related to some other activity. Finally, as in real-life activities all activities have the notion of completeness and start out with this value as false. We leave it to the domain to specify when an activity transitions to the completed status state.

3.2 Operator Ontology

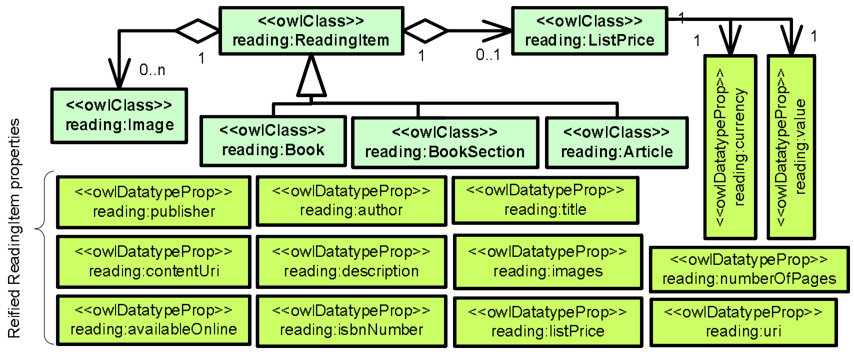
The UAM operator ontology specifies the necessary concepts and relationships for the definitions of the operations on the UAM environment. The operations enable external actors to operate on the UAM environment. We use the operator ontology as input to the generation of our semantic services. The operator ontology defines four primary concepts.

– *uam-op:Operator* represents a particular action on the domain's concepts. Every operator *operatesOn* a domain specific concept.
– *uam-op:Function* represents the active part of the operator. For instance a *Create* function represents operators whose actions result in newly created concept instances in the UAM datastore. Every function also associates with the necessary parameters that it requires.
– *uam-op:OperatorService* combines a series of operator instances into a service. This concept maps one-to-one to a SWS.
– *uam-op:OperatorCode* represents the procedural attachment of the code that gets executed in the UAM datastore when the operator is executed.

4 Demonstration

To demonstrate our UAM SWS, we created an application in a simple domain for which we could discover related SWS on the Web. This showed the feasibility of the system, resulted in a UAM lower ontology for the domain, and an approach to integrate SWS related to the domain. The domain in question is a simplified version of reading group activity. We discussed the primary use case scenario in Section 2.

To create our UAM lower ontology for the domain and annotate available Web services with the domain semantics, we created a *Reading Document* ontology. The main concepts of this ontology are as follows.



<<owlClass>> reading:ReadingItem
<<owlClass>> reading:ListPrice
<<owlClass>> reading:Image
<<owlClass>> reading:Book
<<owlClass>> reading:BookSection
<<owlClass>> reading:Article
<<owlDatatypeProp>> reading:currency
<<owlDatatypeProp>> reading:value
Reified ReadingItem properties
<<owlDatatypeProp>> reading:publisher
<<owlDatatypeProp>> reading:author
<<owlDatatypeProp>> reading:title
<<owlDatatypeProp>> reading:contentUri
<<owlDatatypeProp>> reading:description
<<owlDatatypeProp>> reading:images
<<owlDatatypeProp>> reading:numberOfPages
<<owlDatatypeProp>> reading:availableOnline
<<owlDatatypeProp>> reading:isbnNumber
<<owlDatatypeProp>> reading:listPrice
<<owlDatatypeProp>> reading:uri
0..n
1
1
0..1
1
1
1

Fig. 2. Reading document ontology; showing only the concepts and reified relationships that are of concerns to *Book*-type documents

- *reading:ReadingItem* represents any physical or electronic item that can be read by human agents. This includes Web sites or Web pages as well as some printed materials, e.g., magazines.
- *reading:Book* represents a printed book. This does not include electronic versions (eBook) or audio versions of books. These could be modeled as subclasses of the generic *Book* concept.
- *reading:BookSection* represents a section of a book. This is an important concept in a reading group activity since members of the activity could agree to just read sections of a book (e.g., a page or a chapter).
- *reading:Article* represents a printed article or section of a magazine or a journal. We differentiate Web articles from articles since their properties are typically different. In particular, an article is part of a journal with a volume and issue number, has a publisher, has page numbers, has a title, and has a list of authors.
- *ListPrice* represents the price for the reading item. For a book this list price is the *value* and the *currency* that is usually listed on the back of the book.
- *Image* represents the cover art picture of the reading item (if any).

Figure 2 shows these concepts along with a reified list of the properties and their OWL types. A complete ontology of reading documents would encompass a lot more concepts and some further refinements of the current concepts. We chose to keep the ontology simple to achieve an end-to-end example since we believe the value of our demonstration is in showing how, with limited number of concepts, we can achieve value-add to our activity applications.

The next step in demonstrating our approach and following our use case scenario, is to create a simple UAM lower ontology for the domain. The intent is to define the semantics of reading group activities. Two of the main concepts and properties are: (1) *uam-reading:ReadingArtifact* which is a holder for one reading item. An artifact could be rated, have comments associated with it, relate to other artifacts, be recommended by an actor, and have a reading deadline associated with it; and (2) *uam-reading:ReadingActivity* which represents a reading activity that involves human actors and reading artifacts. A human actor can participate in many reading activities which have a title and a start date. All human actors can be assigned to a reading artifact, comment on them, rate them, recommend them, and relate artifacts to each other.

4.1 Dynamic Discovery and Integration

Since there are not many SWS are currently publicly available, we decided to overlay existing Web services that deal with reading documents with our semantics. We chose the Amazon.com E-Commerce Services (Amazon ECS) since it allows access to the contents from the book department of Amazon's Web site along with the various information collectively gathered from the Amazon community. We created simplified versions of the Amazon ECS specifically exposing capabilities related to our reading document ontology.

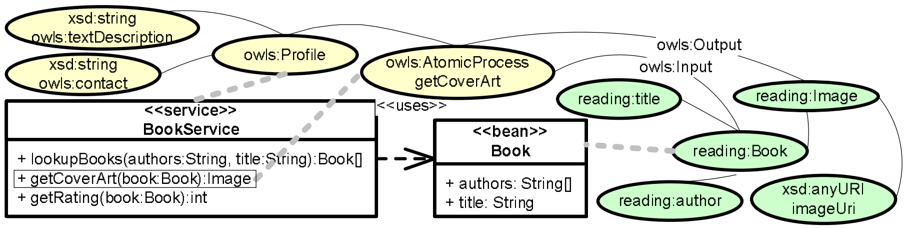


Fig. 3. Simplified *BookService* with partial OWL-S annotation. Heavy dashed gray lines show which part of the service the semantic annotation refers to. The ovals represent the OWL-S concepts and domain-specific annotations.

Figure 3 shows parts of our simplified SWS overlayed with a partial OWL-S descriptions. We only show a subset of the OWL-S *Profile* and portions of the *Process* description for one of the service's methods. The remaining methods would also be described likewise. In addition a *Grounding* instance is also attached to the *Service* instance to point to the WSDL for the service.

Similar to the simplified *BookService* SWS, the generated UAM SWS are overlaid with the appropriate OWL-S descriptions. For instance, we deploy a service to query and retrieve the *ReadingActivity* and *ReadingArtifact* instances, and these have OWL-S descriptions annotated with the *Reading Group Activity* lower ontology and the *Reading Document* ontology. The discovery process is realized by matchmaking the annotations of UAM SWSs with that of the *BookService*. We created a matchmaking agent that runs an algorithm that is similar to [10]. The algorithm looks for *Process* descriptions from SWS for which the *Input* semantically matched the *Output* from the UAM SWS.

Semantic matching either means that the *Input* class is the same as the *Output* class or that the *Output* class is subsumed by the *Input* class, e.g., the *Output* class is a subclass of the *Input* class. A concrete example is to discover that the *BookService getCoverArt* can be passed a *reading:Book* instance from the UAM SWS to generate an *Image* instance which contains the image URI for the cover art. In addition, the matchmaking also looks for cases where the discovery can take multiple *Process* method invocations. For instance, using a *reading:Book* instance *reading:authors* and *reading:title* properties the agent can determine the book's *reading:isbnNumber* which can in turn be used to retrieve the book's *rating*.

5 Future Work

We are constantly expanding the capabilities of our UAM environment. Currently our SWS generation requires the wiring of the operation definition to an existing Java class on the UAM server that operates on the datastore. We would like to eventually bypass this step by having generic Java operators that would operate on different domains and therefore not require specializations when new domains are supported. This could be achieved if the generic operators use the domain ontology as an abstract definition of the operands and the types that are passed as arguments to the operator definitions. We are also looking into using other, simpler, more lightweight SWS approaches, such as WSDL-S [1] as well as expanding our use cases to richer activity domains.

References

1. R. Akkiraju et al. Web Services Semantics: WSDL-S. http://lsdis.cs.uga.edu /library/download/WSDL-S-V1.html, Apr. 2005.
2. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 501(5):28–37, May 2001.
3. P. Dourish. Process Descriptions as Organisational Accounting Devices: The Dual Use of Workflow Technologies. In *Proc. of the ACM Conf. on Supporting Group Work*, Boulder, CO, Sept. 2001.
4. T. Kreifelts, E. Hinrichs, and G. Woetzel. Sharing To-Do Lists with a Distributed Task Manager. In *Proc. of 3rd European Conf. on Computer-Supported Cooperative Work*, Milan, Sept. 1993.
5. D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/, Feb. 2004.
6. S. A. McIlraith, T. C. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, Mar. 2001.
7. T. P. Moran. Activity: Analysis, Design, and Management. In *Proc. from the Symp. on the Foundations of Interaction Design*, pages 12–13, Italy, Nov. 2003.
8. T. P. Moran and A. Cozzi. Unified Activity Management: Supporting People in eBusiness. *Communications of the ACM*, Dec. 2005. To appear.
9. OWL-S. OWL-Service Ontology 1.1. http://www.daml.org/services/owl-s/1.1/, Nov. 2004.
10. K. Sycara et al. Automated Discovery, Interaction, and Composition of Semantic Web Services. *Journal on Web Semantics*, 1(1):27–46, Sept. 2003.