

# Semantic Web Services for Human Activities

E. Michael Maximilien  
IBM Almaden Research Center  
650 Harry Road  
San Jose, CA 95120  
maxim@us.ibm.com

## Abstract

*Semantic Web services (SWSs) extend current Web services standards to help facilitate their usages. While current SWS approaches have shown some early promising results, they have focused on somewhat unrealistic use-cases that make various problematic assumptions and focus on somewhat farfetched usage scenarios. In this paper we present a category of use case scenarios for SWSs that centers around keeping human users in the automation process while facilitating their activities. We demonstrate our approach with a simplified scenario and highlight some of the details of our architecture and implementation. Finally, we also discuss how our approach could be extended and applied in other domains; namely, the domain of asset-based business approach to creating IT computing infrastructure.*

## 1. Introduction

Semantic Web services (SWSs) [17] efforts have focused primarily in trying to automate or semi-automate design and runtime tasks performed by service consumers, e.g., service discovery, service composition, and so on. These approaches assumed that various services are available and annotated with agreed and common ontologies. The semantic annotations form the basis for programming the automation. Naturally, the heterogeneity of the Web and resulting services make this assumption somewhat problematic. In some ways one could argue that the Web's success is directly related to the low barrier of entry to publishing on the Web which is also the cause of its heterogeneity. As a result SWSs technologies have not yet seen widespread deployment in the large or even in enterprise settings.

Furthermore, some of primary use case scenarios for the current SWSs assume that a common ontology is used to represent the needs of service consumers, which can be used to match-make other available SWSs. Additionally, if no available services are found that can meet the needs of

the user, matchmaking mediators can determine if a composed service can be dynamically established by creating a sequence of calls from the available services [23].

We overcome parts of both assumptions of the current SWSs use cases by focusing our approach on a different category of use cases. Instead of automating the design time and runtime tasks of human users of SWSs we focus instead in using SWSs to facilitate and augment human activities [7] while keeping the human agents in the automation process. This results in use cases, we believe, that are closer to widespread adoption, achieves real value, and does not necessitate many readily available and annotated services.

IBM's unified-activity management (UAM) [18, 19] computing environment incorporates the loose and malleable characteristics of human activities by representing activities as first-class OWL [16] instances that are interconnected using a semantic network of relationships representing the context and evolution of the activities. One of the thesis of activity-based systems is that they tend to better reflect actual human work than rigid workflow-based approaches. We implement an activity-based system that includes an upper ontology for activities. The upper ontology defines the basic concepts and relationships for all activities. This includes the concept of *Activity* which associates with other *Activities*, includes *Artifacts*, and involves *Actors*. Domain-specific lower ontologies extend our upper ontology to represent the intricacies that activities typically have in the domain.

For example, an employee appraisal activity lower ontology would contain the concepts of *Achievement*, *Goal*, *Result*, *Impact*, *ApproverActor* and so on; as well as specific relationships such as *quarterlyGoal* to represent the goals of an *EmployeeActor* for the appraisal time span. An employee appraisal activity-based application would allow the human actors to create instances of the domain concepts and interrelate them using the various defined relationships as well as the various work activities that the employee participates in and how the results helped her achieve her goals.

In this paper we investigate the use of SWSs with our ac-

tivity environment. Our primary contributions are in exploring the use of SWSs into a new class of use cases that consider keeping humans in the automation loop. A secondary contribution is in showing the advantages and possible usages of our activity environment’s semantics in the growing world of semantically rich Web-based services on the internet and on the intranet. A final contribution is in creating new classes of SWSs applications resulting in newer incentives for Web service providers to add domain semantics to their services.

## 1.1. Organization

The remainder of this paper is organized as follows. Section 2 illustrates and motivates the usage of SWSs in human-based activity computing environment. Section 3 gives a general overview of our framework. Section 4 presents the details of our demonstration for a simplified domain of activities. Section 5 compares and contrasts related literature on semantic Web services and activity-based computing, with ours. Finally, Section 6 discusses how our approach can be applied to a richer domain and draws some key directions for future work.

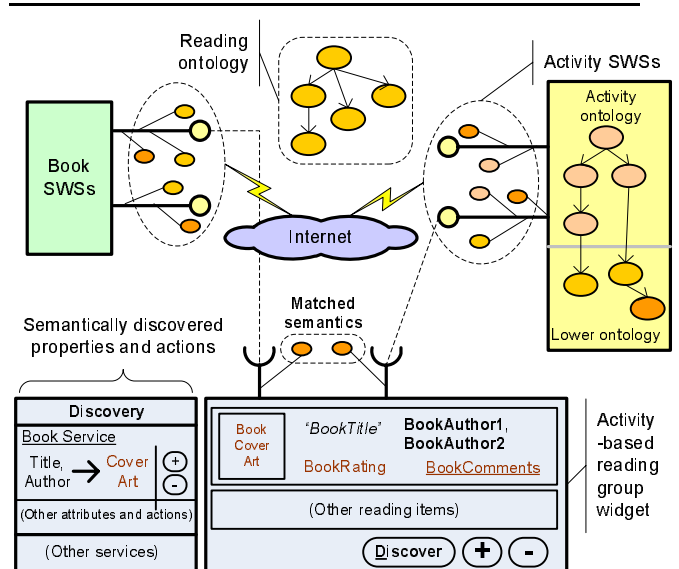
## 2. Motivations

We further motivate our SWS approach to activity-based computing by briefly considering the general motivations for SWS and by discussing the details of a scenario example—which also constitutes a primary use case for our example implementation demonstration in Section 4.

### 2.1. Semantic Web Services Motivated

The Web services standards give a common XML interface and protocol for accessing and invoking services. Services are business functions exposed for remote consumption by Web agents (human or software). While the current set of standards enable interoperability and access to services, their current descriptions lack semantic contents to enable easier automated discovery, selection, and composition. The OWL-Services (OWL-S) [21, 10] proposed service ontology try to overlay semantic descriptions on top of the standard Web services in order to enable some of the aforementioned service usages.

Using OWL-S described services or semantic services, software agents can be programmed to automatically discover services matching a consumer’s service needs. The agents matchmake the consumer’s needs with available semantic services (described using the OWL-S *Profile* sub-ontology). The matchmaking process enables automated discovery of a service’s capabilities. Further, the



**Figure 1. Reading group activity scenario overview; including the reading group activity widget application, the widget’s discovery window, the activity datastore, and an example Book SWS.**

agent could also automatically compose primitive services to create a new composed service that match the consumer’s needs by using the OWL-S *Process* ontology for the primitive services. Sycara et al. [23] describe initial works and results for OWL-S-based SWS.

### 2.2. Scenario: Use Case

In order to demonstrate the use of SWSs into our activity environment, we selected the simple scenario of knowledge workers (or students) sharing common reading activities, e.g., a list of items to read. The point of sharing such activities is to not only have all actors involved in the activity read the items in the list but importantly to share information and maybe uncover insight that otherwise they would not have individually.

Our activity environment can be easily tailored to provide the facilities needed to manage reading group activities. However, since much of the work in a reading group activity, besides the actual readings, will encompass using the Web and making use of available services and sites, such as the Amazon.com Web site, Google searches, and others, we would like to enable semi-automated discovery and integration of these types of services using the semantics of our activities.

Figure 1 shows the various components of our scenario

including a wireframe version of the interface for an end-user activity-based reading activity application widget and its discovery window.

Briefly, our activity-based application uses a lower ontology for reading group activities. This ontology, described in [12], extends the activity upper ontology and uses a *Reading Document* ontology. This ontology includes concepts such as *Book*, *BookSection*, *Article*, *WebArticle*, and so on. We expose OWL-S semantic services that are annotated with our *Reading Group Activity* and *Reading Document* ontologies to allow the creation, modification, and query for the various objects and relationships involved in reading group activities. For example, we have services to find a *Book* artifact in a reading activity.

We create a simple widget activity-based application that gives an interface for the human actors involved in reading group activities. By creating a simplified semantic version of Amazon.com ECommerce Services (Amazon ECS) [1] (which proxies the real Amazon ECS) that we annotate using the *Reading Document* ontology we can automatically discover available cover art pictures and list prices for book reading items as well as Amazon’s community information on the books, e.g., ratings and comments on books. These reading item attributes are discovered by matching the semantics of the activity SWS and the simplified Amazon ECS SWS and can be automatically added, after human user selection, to the activity-based applications.

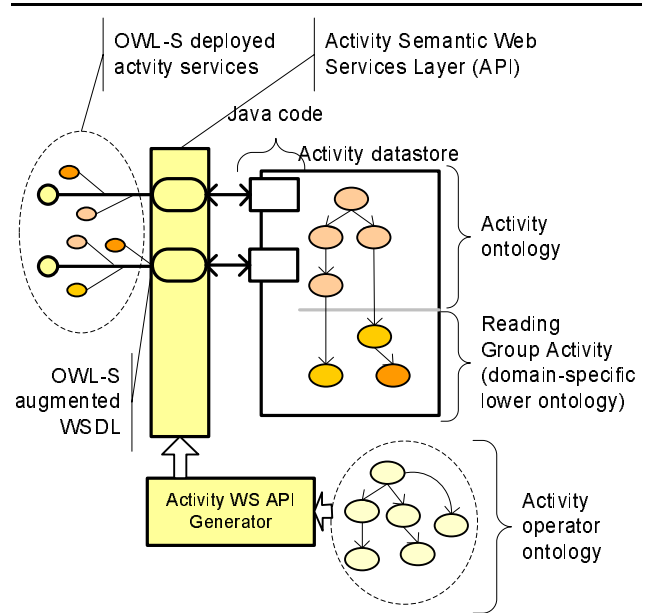
### 3. Framework

We now give a complete overview of our activity-based framework and how we expose a services API to allow the creation of activity-centric applications. We also show in more details how the services are annotated with semantics.

#### 3.1. Architecture

Our activity environment comprises a RDF datastore which keeps OWL instances for all activities, artifacts, actors, and their relationships according to the activity upper and domain specific ontologies [12]. To expose a services API for our activities that maintains the domain-specific semantics, we created an activity operator ontology which allows the definition and generation of Web services representing the operations to *create*, *add*, *modify*, and *find* activity objects.

The services parameters are typed using the domain-specific activity ontology. We maintain the semantics of the domain by creating OWL-S *Profile* and partial *Process* descriptions for the generated services that are annotated with the domain ontology and the domain-specific activity ontology. Figure 2 gives an overview of our architecture.



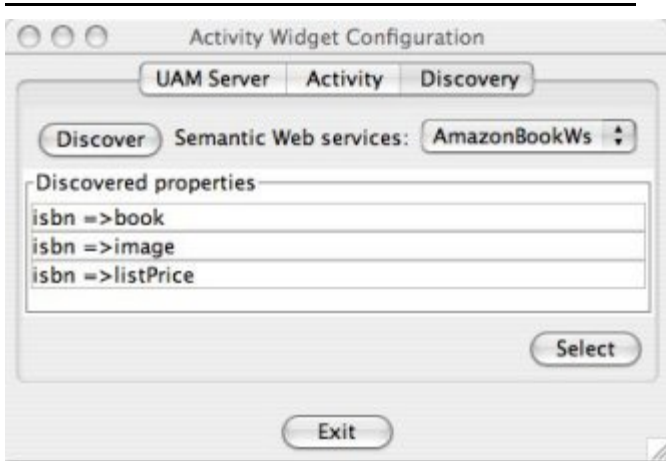
**Figure 2. Architecture overview. The activity datastore contains OWL instances according to the activity upper ontology and the domain ontologies. The activity operator ontology allows the definition of activity SWS for the domain; each generated service is augmented with OWL-S descriptions.**

As an example, for our *Reading Group Activity* ontology we expose SWS with operations to *createReadingActivity()*, *addBookReadingItem()*, *modifyReadingItem()* passing attributes such as author, description, and so on, according to the *Reading Document* and *Reading Group Activity* ontologies.

#### 3.2. Exposing and Consuming SWS

With the activity operations exposed as semantic services, allowing the creation of activity-based applications that maintain the activities’ semantics; we would next want to use the semantics to enrich these applications with services available on the Web. Specifically we want to ease the burden of discovery and integration by making use of the activity semantics. To that aim we exposed the services using the OWL-S ontologies.

For each generated activity SWS, we create an OWL-S *Service* instance (or individual) containing a *Profile* instance, an *AtomicProcess* instance for each method in the generated SWS, and a *Grounding* instance. The *Profile* instance contains information to advertise the service, such as, the contact information and its text description. The *Pro-*



**Figure 3. Screen shot of properties discovered from Amazon ECS SWS.**

cess instances describe the details on how to use each operation of the service. The *Grounding* instance points to the WSDL file for the service which describes the details on how to bind, format, and send SOAP messages to the service.

To understand how the semantics of the activity ontologies are annotated to the generated services it's useful to understand the details of how the *Profile* advertisements and the *Process* instances are created. For each method supported by the service, there is one, or two, messages associated with the service's WSDL: a request and potentially a response message. OWL-S annotation consists of creating a *Process* instance for each message and associating parameter descriptions to each. The parameters are instances of *Input*, *Output*, *Precondition*, and *ResultVar* [11]. Unlike their WSDL counterparts the parameters in OWL-S are described using the appropriate OWL classes from the ontologies where the concepts are defined. Section 4.1.1 illustrates in more details how services are annotated with the semantics defined in OWL-S.

## 4. Demonstration

We now demonstrate our implementation of the scenario discussed in Section 2.2. Figures 3 and 4, show screen shots of the running application.

The first figure shows the discovered properties which can be selected by the user. Figure 4 show the running reading activity application focusing on a Business Reading activity which contains four reading artifacts, each with a book reading item for an actor named Max. Note that each reading artifact contains the discovered and selected properties of book cover art and list price.



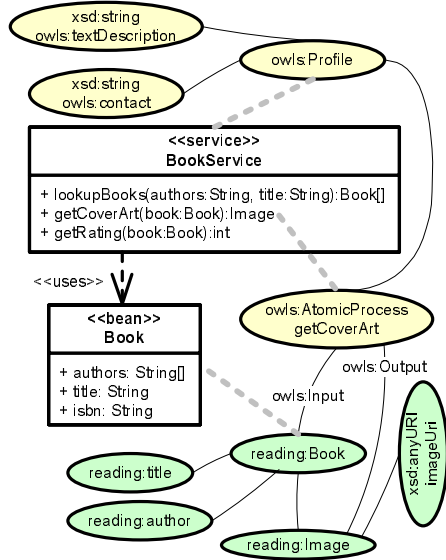
**Figure 4. Screen shot of reading group activity widget showing discovered properties from the Amazon ECS SWS—a proxy SWS to the real Amazon ECS Web service. The images and price list for each book reading item are discovered dynamically from the Amazon ECS SWS.**

### 4.1. Dynamic Discovery and Integration

To discover and integrate properties from services into our domain-specific activity-based application, we needed to have available services in that domain. Since there is a lack of SWS currently publicly available, we took the approach of overlaying an existing Web service that deals with reading documents with our semantics. The Amazon ECS is an obvious choice, since it allows access to the properties from the book department of Amazon's Web site along with the various information gathered from the Amazon community of buyers and sellers.

**4.1.1. Simplified Amazon SWS** We created simplified versions of the Amazon ECS specifically exposing capabilities related to our reading document ontology.

Figure 5 shows parts of our simplified SWS overlaid with a partial OWL-S description. We only show a subset of the *Profile* and portions of the *Process* description for one of



**Figure 5. Simplified *BookService* with partial OWL-S annotation. Heavy dashed gray lines show which part of the service the semantic annotation refers to.**

the service’s methods. The remaining methods would also be described likewise, i.e., with appropriate *Input* and *Output* semantics pointing to the correct concepts and ontologies. In addition a *Grounding* instance is also attached to the *Service* instance to point to the WSDL for the service.

**4.1.2. Semantic Discovery of Properties** Similar to the simplified Amazon SWS, the generated activity 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.

For each service, we create an extended version of a service agent [14] that proxies the service. In a nutshell our service agents are software components with methods reflecting the services they proxy with additional methods and with their own thread of control. These agents are configured to parse the various parts of the OWL-S description for the SWS. In addition, we added capabilities to these agents to help in the dynamic discovery of new properties to enhance the reading group widget.

The discovery process is realized by matchmaking the description of the SWS that the agents’ proxy with the semantics from other SWS. Each agent runs a matchmaking algorithm that is similar to [23]. The algorithm looks for *Process* descriptions from SWS for which the *Input* semantically matched the *Output* from the activity SWS.

## 4.2. SWS Matching Algorithm

Our matching algorithm is sketched in Listings 1 and 2. The algorithms are sketched in Python using a framework which we use to parse the available SWS and ontologies and allows us to operate on various semantics annotations directly as first class objects.

Listing 1 shows the top level of our algorithm. It is called prior to showing the list of discovered properties (Figure 3) to the user. Line 2 assumes that the set of SWS for our reading group activities domain are available. Line 3 is the list of SWS that we need to potentially match with. The for loop at line 4 iterates over each potential matching service and looks to see if it is in the same domain as our application (from variable *self.domainOnto*). We then look for matching properties that are obtained from matching the the outputs of the *dServices*.

Listing 1: Matching algorithm. Matches the reading group activity SWS with discovered SWS in the same domain.

```

1 #...
  dServices = ... # list of domain services
3 services = ... # get from registry of SWS (
  currently fixed)
  for s in services:
5     if s.owl.profile.sParameter.uri == self.
      domainOnto:
      props = self.matchProperties( s,
      dServices )
7     self.displayProps( props ) # displays
      DiscoveredProp
#...

```

Listing 2 shows the details of our matching property algorithm. It consists of three procedures:

**matchProperties.** Iterates over all domain services and looks for matching properties.

**findMatchingProperties.** For each *atomicProcess* in the service *s* we try to match its inputs with the outputs of the domain service. Note that the output are decomposed to reduce them to individual elements of the ontology (where possible) in order to increase the likelihood of a match. Line 23 iterates over each output and looks for a potential match with one of the input (calling the *matchInsOut* function of line 32). If one exists the list of input is reduced in line 25 and collected in line 26. If all inputs are matched then a property is discovered and thus created and added to the map that will be returned.

**matchInsOut.** The matching of inputs and output occurs by iterating over all inputs and seeing if the output class is the same or is subsubmed by the input class (line 34).

Listing 2: Properties matching algorithm. Matches properties that can be generated by using the outputs from one SWS to the inputs of another.

```

10 # @return a map of matches properties for
    discovered services
    def matchProperties( self , service , dServices ) :
12     props = {}
        for s in dServices :
14         self.findMatchingProperties( service , s ,
            props )
        return props
16
    # @return a map of discovered properties object
18 def findMatchingProperties( self , service ,
    dService , props ) :
        for atomicProcess in service.atomicProcesses :
20         matched = 0
            inputs = atomicProcess.inputs
22         params = []
            for output in dService.profile.outputs : #
                decomposed output
24             if self.matchInsOut( inputs , output ) :
                    inputs.remove( output )
26                 params.append( output )
                    if len( inputs ) == 0 :
28                     dProp = DiscoveredProp( atomicProcess
                        , params , dService )
                        props[ atomicProcess.name ] = dProp
30
    # @return true (1) if there is one input matching
        the output
32 def matchInsOut( self , ins , out ) :
        for i in ins :
34             if out.class == i.class or i.class.subsumes
                ( out.class ) :
                    return 1
36     return 0

```

## 5. Related Work

We divide the works related to ours into two categories: semantic Web services and activity-based computing. For each category, we compare and contrast relevant and recent works that are closest to ours.

### 5.1. Semantic Web Services

Adding semantics to Web services has received a lot of attention in the Semantic Web, Web services, and multi-agent systems (MAS) communities. Initial works from the DAML-S (now OWL-S) coalition laid the groundwork for

most of the current SWS ideas [3, 17]. As previously mentioned, our matchmaking algorithm is based on the algorithm by Sycara et al. [23]. However, our purpose is not to create a composed service but rather to discover new properties that we allow the user to select and enhance the widget application.

Akkiraju et al. [2] propose a lightweight approach to adding semantics to Web services called WSDL-S. In their approach, the semantics of the service's operations are directly added to the WSDL file. This facilitates the deployment as well as usage. Since we do not use the full features of the OWL-S *Process* sub-ontology, we believe that WSDL-S may be a promising simplification for our application and purposes.

The Web Services Modelling Ontology (WSMO) [24] provides a comprehensive framework for SWS and takes a mediation approach for data and process by eliciting goals from the users of their environment. Two key differences with ours are that: (1) we focus on human-based activities and (2) we do not require our agents to explicate their goals.

Recent work by Sheshagiri et al. [22] describe using SWS to support mobile-based applications. They take an agent approach, to discover and integrate available services in the mobile environments. However, unlike our work, they do not try to facilitate any types of activities and currently constrained themselves to mobile contexts and devices.

Our service agents continue our own previous work in overlaying services with agents that are programmed to automate some aspects of the service consumption [13, 14]. In particular, the service agents facilitate the parsing of the service's OWL-S semantics and helping the discovery of new properties.

### 5.2. Activity-Based Computing

Our current work on adding SWS to activity-based computing extend previous works on activity described in [18] and [19]. Muller et al. [20] describe an activity explorer application which is also part of IBM's UAM initiative and has influenced our work.

Initial work on activity-based computing by Kreifelts et al. [9] represent activity-based computing as distributed shared tasks. IBM's UAM supports the distributed shared tasks concepts but additionally adds semantics to the activities which in return enables the type of dynamic discovery processes described in this paper.

Kaptelinin, Bardam, and Dragunov et al. [8, 5, 6] are other activity-based systems. They capture low-level user events and try to recognized them as clustering into higher-level activities. They differ from our efforts in that we allow activities to be created from simpler events as they do but also directly at the higher level notion of human activi-

ties. This flexibility is mainly due to the semantic representation of activities using OWL and RDF.

Finally, none of the activity-based systems that we have reviewed looked into merging the worlds of activities with SWSs. Since the semantics of our activities use the same language as the Semantic Web and the efforts in SWS, we are able to use the same ontologies which may result in easier dynamic integration.

## 6. Discussion

Our environment is a comprehensive system that enables activity-based computing. Essentially, our environment is a semantic datastore of interrelated activity instances that represent the loose and dynamic structures of real-life activities. To enable the creation of a variety of activity-based applications, especially applications that are specific to a domain, we expose an extensible ontology and a remote API that maintains the semantics of the activities.

The activity ontology is designed to be extended with domain-specific ontologies (or lower ontologies) that encompass the concepts and relationships in the domain. In this paper we presented how a simple domain-specific lower ontology can be added and used.

To enable the creation of external applications, we exposed a Web services API that maintains the semantics of the activities involved. The exposed API is customizable in order to account for the domain-specific concepts. We achieve this by defining an ontology for the exposed SWS. In addition, using the lower ontology, we generate OWL-S annotations for each of the SWS with the appropriate grounding information pointing to the generated activity Web services' WSDLs [11].

To show the value of the activity semantics and maintaining the semantics in the exposed services, we created a simple domain-specific application. Our domain is reading group activities and our application allows a group of human actors to view, share, and manage their reading activities. Using the exposed semantics in the activity SWS we were able to discover and integrate new information about the reading artifacts from external services that use a common ontology as our *Reading Group Activity* lower ontology.

### 6.1. Asset-Based Business Activities

Although our demonstration used a simple domain and a simple activity application, it can be extended for a more complex domain that involves many activities with different actors and various artifacts. One such example is the domain of asset-based IT businesses.

Current IT service businesses are going through a paradigm shift. They are transitioning from a human-

intensive business to an asset-based business [4]. In this new business environment assets (e.g., computer software and hardware, software byproducts, software libraries, and so on) constitute an important aspect of the capital value of the business. Therefore, managing, sharing, using, and reusing assets become important day-to-day activities for service engineers. We posit that such activities can benefit from our activity environment and the activity SWS.

Similar to the *Reading Document* ontology we can imagine the creation of an *Asset* ontology defining the concepts within the domain and their relationships. For example, the notion of an *Asset* would be specialized with subclasses such as *HardwareAsset*, *SoftwareAsset*, and *DocumentAsset*. Every asset would have relationships such as *authors*, *lastUsed*, *usageHistory*, *ratings*, *relatedTo*, and *requires*. These relationships could then be further extended and additional relationships created for the specialized asset types.

Assuming the IT services business would keep a datastore of the various assets it owns, we could then create a domain-specific activity ontology for asset activities and build applications for adding, searching, and sharing asset-related activities. Similar to the reading activity scenario, an equivalent scenario for using the assets would be to discover, within the intranet and Internet, asset-related services.

For instance, let's assume that within the organization, groups of service engineers that create and use software assets, also collaborate to share their opinions on these assets. The opinions would include comments but also links to related assets, inside and outside the organization. For example, links to open-source projects and online documentations, Web articles, and code snippets that show how the software asset can be further reused and applied. Building the applications and exposing services as we have demonstrate in Section 4 would enable their dynamic discovery and integration as part of activity-based applications.

To illustrate this point, let's assume that sites like SourceForge.net and O'Reilly.com, would respectively expose their software assets and software documentations and discussion assets annotated with the common *Asset* ontology. Then by exposing their functions as SWS, our activity-based applications would be able to discover and integrate the new functionalities and knowledge dynamically and enrich the capabilities and experiences of service engineers. This in turn would potentially lead to more effective usages and sharing of assets and therefore add value to the business.

### 6.2. Future Work

Supporting new domains such as the asset-based business discussed in Section 6.1 would allow us to validate our

SWS environment and also allow opportunities for the creation of other rich activity-based applications in different contexts; which indirectly would help us further validate our activity-based environment.

Another interesting direction, that helps address the challenges of requiring a common ontology, is to take advantage of the social networking aspects of our activity environment. Since users of particular activity-based applications participate in the same domain, we could program our applications to allow these human agents to annotate (using the available semantics and using freeform tags) the SWSs or other artifacts of the activities. This would allow us to potentially create a *folksonomy* [15] for the different domains which can then be refined and used to annotate services and our activities.

Finally, an important direction for future work is in the discovery and integration of SWS. Currently the number of publicly available Web services that are annotated with semantics is small. An incentive for businesses to spend the time moving to a Service-Oriented Architecture and then adding semantics to their services would be to show the gains of such efforts, e.g., dynamic discovery, integration, and selection of these services by using the semantics. Additional results in this line of research that relate to the business values gained would go a long way to encourage the creation of SWS and applications, thereby furthering the vision of the Semantic Web.

## 7. Acknowledgment

We thank the IBM Research UAM teams in Cambridge, MA as well as San Jose, CA for their efforts, contributions, and their passionate discussions.

## References

- [1] Amazon E-Commerce Service Developer Guide. June 2005, Version 4.0.
- [2] R. Akkiraju, et al. Web Services Semantics: WSDL-S. <http://lsdis.cs.uga.edu/library/download/WSDL-S-V1.html>, Apr. 2005.
- [3] A. Ankolekar, et al. DAML-S: Semantic Markup for Web Services. In *Proc. of the Intl Semantic Web Working Symposium (SWWS)*, pages 411–430, July 2001.
- [4] F. Barber and R. Strack. The Surprising Economics of a “People Business”. *Harvard Business Review*, 83(6), June 2005.
- [5] J. E. Bardram. Activity-Based Computing: Support for Mobility and Collaboration in Ubiquitous Computing. In *Proc. of Personal and Ubiquitous Computing*, July 2005.
- [6] A. N. Dragunov, et al. TaskTracer: A Desktop Environment to Support Multi-tasking Knowledge Workers. In *Proc. of the Conf. on Intelligent User Interfaces*, San Diego, CA, Jan. 2005.
- [7] D. C. Engelbart. Augmenting Human Intellect: A Conceptual Framework. SRI International TR AFOSR-3233, October 1962.
- [8] V. Kaptelinin. UMEA: Translating Interaction Histories Into Project Contexts. In *Proc. of Conf. on Human Factors in Computing Systems (CHI 2003)*, pages 353–360, Fort Lauderdale, FL, Apr. 2003.
- [9] 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.
- [10] D. Martin, et al. OWL-S: Semantic Markup for Web Services. <http://www.daml.org/services/owl-s/1.1/overview>, 2005.
- [11] D. Martin, et al. Describing Web Services Using OWL-S and WSDL. <http://www.daml.org/services/owl-s/1.1/>, Nov. 2004.
- [12] E. M. Maximilien, A. Cozzi, and T. P. Moran. Semantic Web Services for Activity-Based Computing. In *Proc. of IC-SOC’2005*, LNCS 3826, pages 558–563, Germany, 2005.
- [13] E. M. Maximilien and M. P. Singh. A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing*, 8(5):84–93, Sept. 2004.
- [14] E. M. Maximilien and M. P. Singh. Toward Autonomic Web Services Trust and Selection. In *Proc. of 2nd Intl Conf. on Service Oriented Computing (ICSOC)*, pages 212–221, New York, Nov. 2004.
- [15] P. Mika. Ontologies Are Us: A Unified Model of Social Networks and Semantics. In *Proc. of 4th Intl Semantic Web Conf. (ISWC)*, LCNS 3729, Springer-Verlag, 2005.
- [16] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. <http://www.w3.org/TR/owl-features/>, Feb. 2004. W3C recommendation.
- [17] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, Mar. 2001.
- [18] T. P. Moran. Activity: Analysis, Design, and Management. In *Proc. of the Symposium on the Foundations of Interaction Design (IVREA)*, pages 12–13, Italy, Nov. 2003.
- [19] T. P. Moran and A. Cozzi. Unified Activity Management: Supporting People in eBusiness. *Communications of the ACM*, Dec. 2005.
- [20] M. Muller, et al. One Hundred Days in an Activity-Centric Collaboration Environment based on Shared Objects. In *Proc. of Conf. on Human Factors in Computing Systems (CHI 2004)*, Vienna, Apr. 2004.
- [21] OWL-S. OWL-Service Ontology 1.1. <http://www.daml.org/services/owl-s/1.1/>, Nov. 2004.
- [22] M. Sheshagiri, N. Sadeh, and F. Gandon. Using Semantic Web Services for Context-Aware Mobile Applications. In *Proc. of MobiSys2004 Workshop on Context Awareness*, Boston, June 2004.
- [23] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated Discovery, Interaction, and Composition of Semantic Web Services. *Journal on Web Semantics*, 1(1):27–46, Sept. 2003.
- [24] WSMO. Web Service Modeling Ontology. <http://www.wsmo.org/>.