Editor: James Hendler
University of Maryland
hendler@cs.umd.edu

Ontology-Enabled Pervasive Computing Applications

Ryusuke Masuoka and Yannis Labrou, *Fujitsu Laboratories of America*
Bijan Parsia and Evren Sirin, *MIND Lab, University of Maryland*

Information technology's rapid evolution has made tremendous amounts of information and services available at our fingertips. However, we still face the frustration of trying to do simple things in the device- and application-rich environments where we live and work. Let's consider the ritual of visiting a lab or a business unit to deliver a presentation and collaborate with colleagues. We exchange printed business cards that end up in a desk drawer because we never have the time to enter them into our personal information manager. We must deal with capricious projectors that might not work as intended while we make a presentation. We promise to email a soft copy of our presentation when we get back to our office or follow up with a new scheduled meeting. We search for addresses and directions to a restaurant or the airport and print them on paper to take with us. Performing each of these simple tasks entails a complex choreography of interactions with a variety of devices and applications, and manual cutting and pasting, resulting in time-consuming, error-prone, frustrating processes.

Fujitsu Laboratories of America and the MINDSWAP research group at the University of Maryland have created an environment operating in a conference room where all the tasks just described and more can be done with just a few simple point-and-click operations. More than 30 types of services such as "view on projector," "print," "map," and "play (video)" are in the conference room and on the local network—services that you can use and manipulate in many ways. For example, you can easily view and control your local presentation file on any viewing service in the room (for example, "view on projector") with only mouse clicks through your Web browser. If you install our software environment on your computer, you can do even more powerful things with your computer's local services combined with services available pervasively and remotely.

Our research aims to empower nonexpert users with the ability to perform complex tasks in information-rich, device-rich, and service-rich environments using devices (limited in Graphical User Interface real estate, computational power, and so on). Our approach is to expose the functionality in such environments (device functionality or third-party functionality) as Semantic Web services, which in turn the user can discover and arbitrarily compose. We call this approach *task computing*, which we define as computation to fill the gap between the tasks that users want to perform and the services that constitute available actionable functionality.[1]

STEER in TCE

To support task computing, we have implemented a Task Computing Environment including client environment, service discovery mechanism, and Semantic Web services and tools. TCE is composed of several components including STEER (Semantic Task Execution EditoR), White Hole, and PIPE (Pervasive Instance Provision Environment) shown in Figure 1.

STEER is a Task Computing client, which has a Web page interface (shown here) as well as a Java interface. STEER's Compose page is shown in the Web browser. The possible service compositions are categorized based on their semantic inputs and outputs. In each pair, any service in the right-side drop-down menu can be combined with any service on the left-side drop-down menu. The user chooses the service from the drop-down menu and executes the composition. The user clicks the construct button to create a more complex composition starting from the two-service composition in this Compose page. The swirl on the upper right corner is the "White Hole." The user can drag and drop objects such as files, URLs, contacts, and schedules from PIM to create services that provide semantic versions of the objects dropped into it. On the bottom is a service management GUI by PIPE. Through this GUI, the user can make the services local or pervasive or temporarily hold the services for possible future use. The user can also control how services are provided such as expiration time, and so on.

STEER lets fairly unsophisticated users find, select, compose, and execute Web services to achieve common tasks. While STEER typically has several built-in or local services plus a relevant set of remote services on the Web, it

1094-7167/03/$17.00 © 2003 IEEE
Published by the IEEE Computer Society

becomes especially interesting when it can work with a rich, varied set of pervasive services (many of them on devices and peripherals).

By exposing the various functionalities as Web services and advertising them via Universal Plug and Play (UPnP, www.upnp.org), STEER clients on portable devices dynamically discover the services available in the room where they are physically present. Tying service discovery to physical presence both filters the set of services to a manageable level (after all, one generally doesn't care about projectors in rooms on another floor) and suggests some organizing principles for those services. The room's purpose largely determines the sorts of tasks facing users when in the room. The actual devices installed further constrain the set of possible and probable tasks. Finally, the Web services' semantic descriptions provided by the devices permit STEER to automatically combine services into likely compositions for the end user to consider, select, extend, and perform. STEER bridges the gap between what users want to do and what they can do by organizing the unwieldy mass of services into likely courses of useful action. Moreover, STEER helps users more clearly understand what they need and want to do by making likely and common compositions obvious to users.

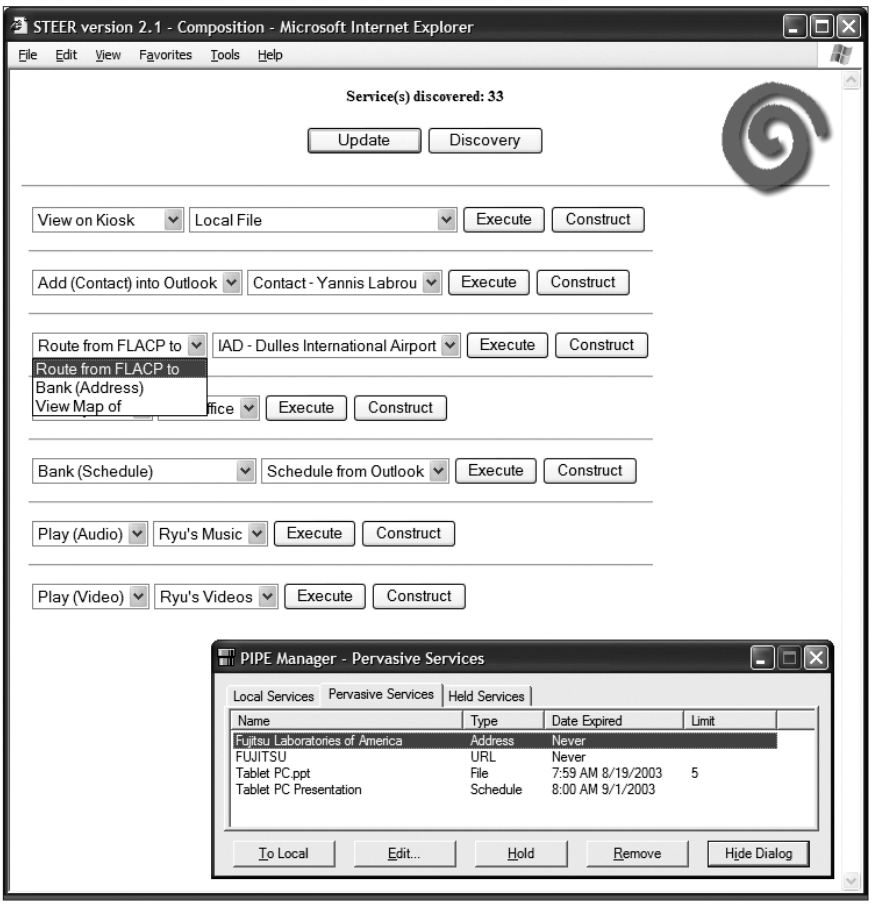


Figure 1. Screenshot of the Task Computing Environment client desktop.

So you want to do what … with Web services?

The type of Web service composition that STEER supports is a simple form of programming, and programming, even a simple form, is not commonly done nor done well by end users. While a simple macro recorder can follow users' actions in performing a task and play those actions back at a later time, such recording requires users to already know how to perform the task. Task computing presumes that users tend to not know in advance how to achieve their goals or even what those goals are. STEER gives users some starter compositions and helps guide them in extending them. This guidance takes two fundamental forms: documentation for end users and suggested compositions and composition steps from STEER. When building a composition, relatively few services are compatible with the prior steps. STEER shows users only the sensible next steps, plus the associated documentation. Faced with limited options plus more detailed explanation as desired, users generally have an easy time choosing what to do. To support user-guided composition, STEER must find sufficiently rich descriptions of the services.

Most Web services are described by Web Service Description Language documents.[2] WSDL is a nice, simple Interface Description Language well suited for developers and developer-class tools. WSDL is easy to integrate with popular programming languages such as C#, Java, and Perl, and is no more difficult to read, understand, or use than the typical module system. Indeed, given its simplicity, it's far easier than most. If you have a clear idea of what you are trying to accomplish and a good grasp of programming, WSDL makes composing and invoking Web services trivial. What WSDL doesn't provide is extensive, structured documentation of the service, nor does it easily support automated composition. Universal Description, Discovery and Integration of Web Services (www.uddi.org) tries to cover some of this gap, but it targets developers, not end users. DAML-S[3] and its successor, OWL-S, supply ontologies for describing Web services so that they might be discovered, explained, composed, and executed. OWL, the Web Ontology Language,[4] extends the Resource Description Language with powerful modeling constructs sufficient for naturally describing many subject domains. Built on the metadata facilities of Resource Description Framework (RDF), OWL effectively describes all manner of Web resources for both human beings and programs. This is, of course, exactly what we needed for STEER.

Although OWL-S provides the means for describing Web services, it doesn't dispense with WSDL. WSDL remains the basic description of Web services. OWL-S connects to WSDL by grounding its descriptions in the lower-level WSDL descriptions. Roughly, WSDL provides information on how to invoke a Web service and OWL-S lets us say what the service does, why we might want to use it, and what to expect if we use it, among other things. Even though there is an expressive gap, OWL-S makes

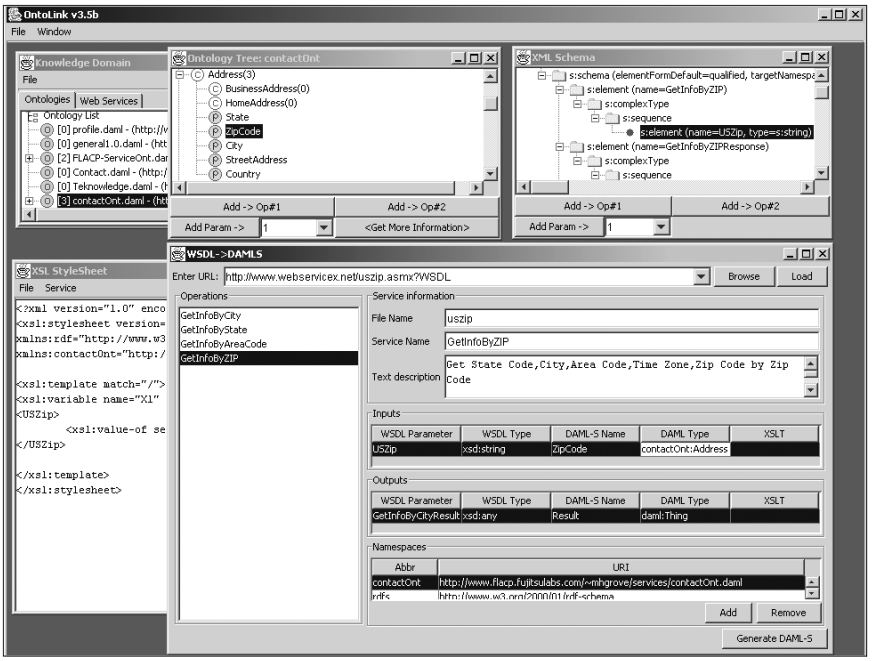


Figure 2. OntoLink is used to generate an OWL-S description from a WSDL annotation for grounding.

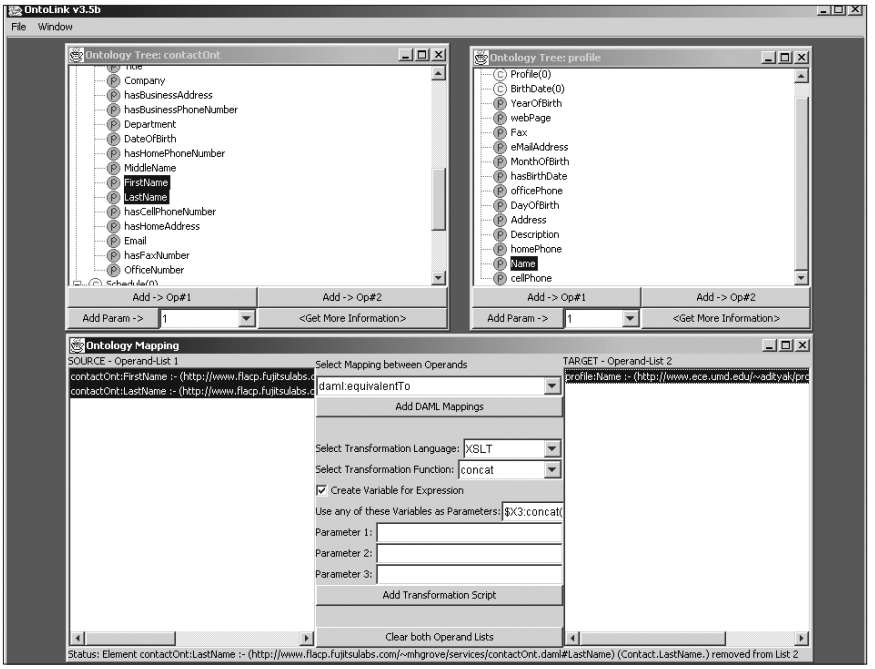


Figure 3. OntoLink for semantic object mapping. OntoLink defines a transformation function between two concepts from different ontologies. This mapping information is described as a Web service and used in compositions to connect two otherwise incompatible services.

good use of what WSDL offers. OWL-S descriptions include a grounding specification that defines how an OWL-S service is bound to a WSDL service so the information in WSDL can be used for invocation. (This relationship should get even smoother with WSDL 1.2, which will contain a canonical mapping into RDF and OWL.)

But there's still a gap between a WSDL description and corresponding OWL-S descriptions. We often exploit that gap to present the base Web service functionality in many ways. But the gap still needs filling. In particular, WSDL documents tend to describe input and output parameter types in terms of XML Schema. While certainly adequate for programmers, and although OWL is perfectly capable of handling XML Schema types, they aren't the best things for naïve end users and they aren't the easiest things to automatically compose. As an example, a WSDL description states that a service accepts a floating number—not very useful information. But mapping this type to a price concept, which is defined in an OWL ontology and linked to other concepts such as currency, dollars, and so on, lets users and software agents better interpret the meaning.

WSDL's extraordinary tool support is one of its compelling features. In many IDEs, WSDL generation and consumption is a simple touch of a button. For the STEER user, OWL-S descriptions are even more transparent. But if we are to expect Web service providers or third-party Semantic Web service description providers to add the extra markup that OWL-S permits, we must give them tools that make the job feasible.

OntoLink

OntoLink lets developers generate OWL-S descriptions from WSDL annotations (see Figure 2).The OntoLink tool's main function is to ground the semantic service described in an OWL-S description to its corresponding Web Service implementation. The mapping between XML Schema types and OWL ontologies is defined and saved in OWL-S's grounding information.

When a WSDL file is loaded, the program automatically generates a simple semantic description template by using the information in WSDL, such as text descriptions, parameter names and types, and so on. A simple point-and-click interface lets users extend this template by defining mappings between the XML Schema types and OWL concepts. For example, a WSDL description could define a parameter to be a string of five digits. Users select the *Address* concept from the class hierarchy, link the parameter to the *ZipCode* property, and state that the *Country* property of *Address* should equal USA (italic words represent concepts defined in OWL ontologies).The transformation functions generated here are then stored as XSLT scripts and put inside the grounding specification

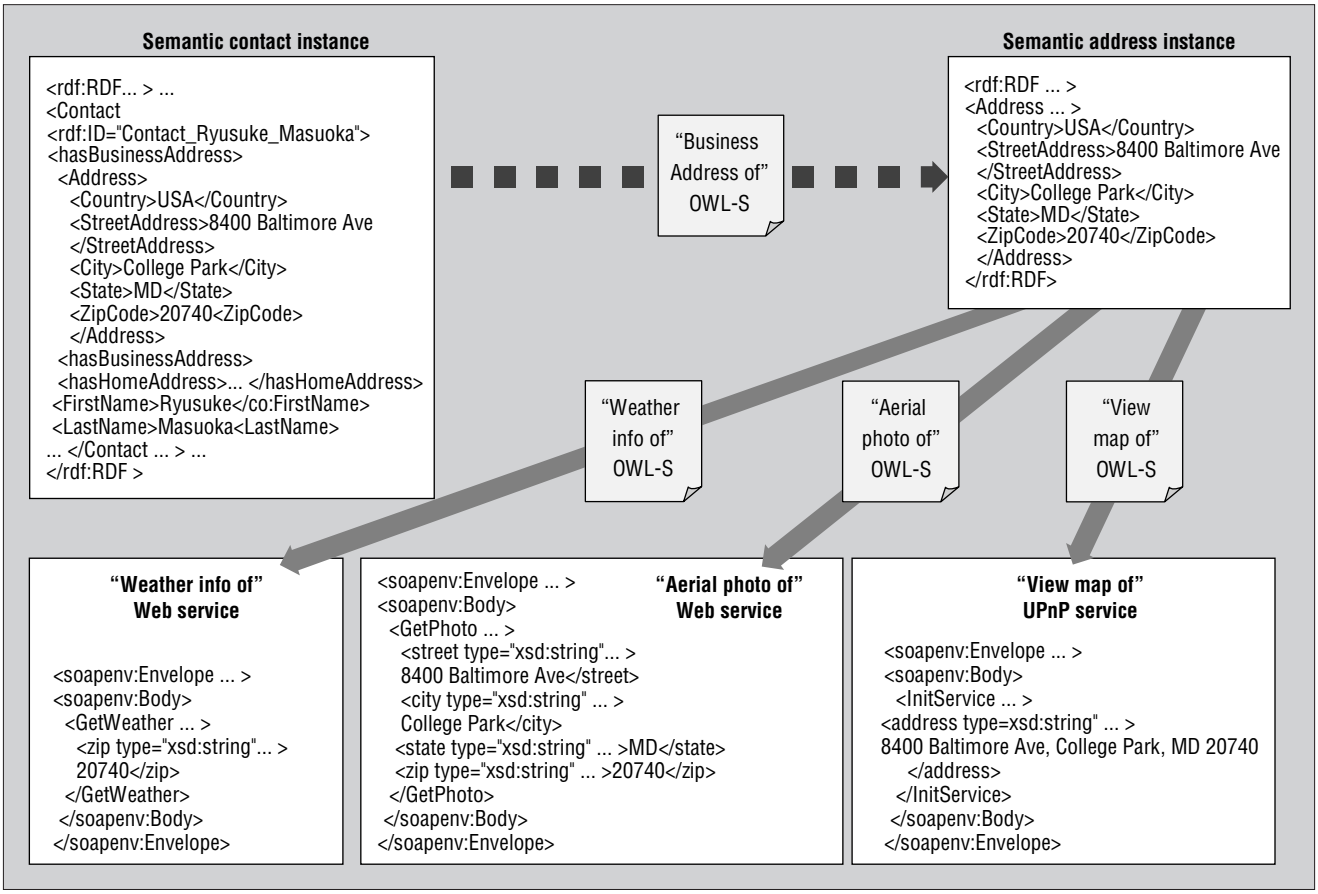


Figure 4. Semantic object mapping and groundings by OWL-S used in STEER. All the XML data is taken from actual executions of service compositions and edited for readability.

of the OWL-S description. Easily creating semantic descriptions for WSDL services makes it possible for STEER to access and invoke Web services on the Internet. As a result, STEER provides a richer environment where users can combine pervasive services with the ones available on the Web.

So far, we've glossed over the problem of service inputs and outputs belonging to different ontologies. In fact, we definitely do not expect the world of specific pervasive environments to be accounted for by a single universal ontology—we consider this idea unrealistic. One large, common ontology is difficult to create, agree on, and manage. Our idea is to start with small ontologies for specific domains and purposes: Only when interoperability between those domains and purposes becomes necessary should we create mappings between (only) necessary parts of them. Mapping doesn't have to be complete or exact; it is good enough as long as the mapping serves the purpose reasonably.

You can also use the OntoLink tool to define these mappings between ontology elements (see Figure 3). Using the same interface as in the XML Schema-OWL mapping, we can define mappings such as a *Name* property of a *Person* concept defined in ontology A is the concatenation of *FirstName* and *LastName* properties that are defined in ontology B. Even arithmetic computations between ontology elements, such as converting the value of the *Length* property from Inches to Meters, can be integrated into these transformations, as XSLT provides quite expressive functions.

In our framework, the semantic object mapping or transformation functions defined between ontology elements are also published as Web services. We can compose two services that use disjoint ontologies by simply adding the appropriate transformation service in between. This approach lets us handle the ontology mapping in a simple and consistent system. STEER integrates the transformation services with the compositions automatically as needed so the user is presented with a set of services that can be seamlessly composed together.

Figure 4 shows how STEER actually uses semantic object mappings and groundings by OWL-S descriptions generated by OntoLink. The "Business Address of" OWL-S file provides a semantic object mapping via its XSLT script, thus making it possible to connect contact-producing services with address-consuming services. XSLT scripts in OWL-S files for address-consuming services marshal semantic address instances into appropriate SOAP messages for invocations of Web and UPnP services. Therefore, a "Contact Providing" service can be composed with, for example, "View Map of" service through "Business Address of" (semantic object mapping) service, and the composition can be executed using only XSLT scripts in these OWL-S files, to show the area map around the business address of the contact.

Why ontology and pervasive computing … at all?

The ad hoc, spontaneous, and dynamic nature of the pervasive computing environment requires the systems to be late-binding. The user interface, available while on the go, is usually limited in modalities, bandwidth between users, and so on. Ontologies describing services give the system the necessary semantics to provide users with enough functionality in limited user interfaces without really accessing services themselves.

Ontologies in the pervasive computing environment are more manageable compared to, for example, those for the Internet. Ontologies for devices will be created by device manufacturers, which can put resources into their creation. They can even create an oligopoly of ontologies by forming consortia. Embodiments of devices with physical representations related to the particular location lead to simpler ontologies. You can have the same device in the next room or downstairs, and there is real reuse of ontologies enabled by natural boundaries in physical environments.

On the other hand, people and companies on the Internet are under the constant pressure of differentiating from others because of the Internet's universal connectivity (the very reason for its success). Even end users are pressed to present views of the world that are different from others. These differences are very difficult for ontologies to capture.

Our implementation of a rich, user-friendly task computing environment depends crucially on Web ontologies written in the OWL language. In our observation, OWL provided a reasonable balance between power and accessibility. Although we were happy that both OWL and OWL-S have their advanced capabilities, we didn't really use them to the fullest possible degree. It turns out that a little semantics goes a long way.

Or so we believe. While very loosely coupled, the components of our task-computing environment are based on a set of technologies designed to work well together. Each part has its exciting moment, but it's the way they mesh together that makes working with the system natural and compelling. It seems possible that the system's overall feel could be preserved even if we relied less on ontologies.

In a sense, we do rely less on ontologies as traditionally conceived and used. The user's understanding of the functionalities available in the environment were as much determined by extra-ontological metadata as by the formal relationships between terms. For example, we found comments and term labels embedded in the ontology invaluable. The formal description allowed STEER to put the service in the right place, but it's the human-centered descriptions that give the user confidence to make the final selection. Similarly, connecting high-level ontological descriptions with lower-level XML Schema and WSDL descriptions not only allowed STEER to automatically invoke composed services but made those services comprehensible to nonprogrammers.

So, while ontologies are key, how they fit in with the wide variety of representational schemes present in our increasingly heterogeneous but interconnected information systems will make or break them. By taking to heart the Web's lessons, ontologies stand poised to break free from their historical niche to become an important and valued part of our everyday computing experience. Both ontologies and our computing experience will be the better for it. ■

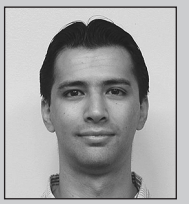
References

1. R. Masuoka, B. Parsia and Y. Labrou, "Task Computing—The Semantic Web Meets Pervasive Computing," *Proc. 2nd Int'l Semantic Web Conf. 2003* (ISWC 03), Springer-Verlag, 2003, to be published.
2. W3C Web Services Description Working Group, www.w3.org/2002/ws/desc.
3. S. McIlraith and D. Martin, "Bringing Semantics to Web Services," *IEEE Intelligent Systems*, vol. 18, no. 1, Jan./Feb. 2003, pp. 90–93.
4. W3C Web-Ontology (WebOnt) Working Group, www.w3.org/2001/sw/WebOnt.

Ryusuke Masuoka is a senior researcher with Fujitsu Laboratories of America. Contact him at rmasuoka@fla.fujitsu.com.

Yannis Labrou is a researcher with Fujitsu Laboratories of America. Contact him at yannis@fla.fujitsu.com.

Bijan Parsia is a research philosopher at the University of Maryland's MIND Lab. Contact him at bparsia@isr.umd.edu.



Evren Sirin is a PhD student at the Computer Science Department of University of Maryland, College Park. Contact him at evren@cs.umd.edu.