The ubiquity of Internet standards such as TCP/IP, HTTP, and HTML has brought the concept of a global cyberspace village closer to reality. The Extensible Markup Language (XML), an evolutionary representation format for interoperability, has become the preferred alternative for exchanging messages among heterogeneous systems. Additionally, emerging schema and semantic standards—including document type definition, the resource description framework (RDF), and the DARPA Agent Markup Language and Ontology Inference Layer (DAML+OIL)—provide tools for describing Web-accessible computational resources in terms of machine-readable metadata.

The recent adoption of XML-based Web services standards has further integrated distributed resources with open Internet protocols at the software service level. Developers are moving beyond simply extending Web resource access and are making a sustained effort to tap into the depth of mutual understanding that can be culled from the Web’s diverse content. Ultimately, they will organize this information conceptually and place it in the semantic Web’s framework.

For end users, the vast increase in available services these initiatives provide can imply either more freedom or more chaos. This flood of options raises the vital issue of how emerging technologies can help to automate the manual operations individual applications require. To meet this challenge, we built our smart office task automation framework using Web services, an ontology, and agent components. SOTA thus provides an integrated information service platform that offers user-centric support for automating intranet office tasks.

PERTINENT TECHNOLOGIES

We drew upon several technologies to create the SOTA platform.

Web services

In SOTA, Web services define programmable application interfaces logically accessible using standard Internet protocols. Applications access these services using ubiquitous Web protocols and data formats such as HTTP and XML, defined through the simple object access protocol (SOAP). SOTA uses the Web Services Description Language (WSDL) to define and describe the programming interface for the SOAP messaging service’s independent implementation. The universal description, discovery, and integration (UDDI) specification serves as a centralized services registry that offers a global services market.

Although a service requester can browse standard service categories to search for services or use a keyword query to obtain the services via UDDI, the
absence of a semantic description for the service operation results in low recall and yields imprecise results. Recently, researchers have combined semantic technology with the inference, matchmaking, and dynamic composition of Web services.

Ontology

Many researchers consider an ontology to be a theory of content that pertains to object types, properties, and possible relationships in a specified knowledge domain. In the context of our work, an ontology is the explicit formal specification of terms in a domain and the relationships among them. Intelligent applications can use a shared ontology to identify unified meaning, then solve a problem or infer an outcome.

The draft American National Standard Knowledge Interchange Format is a frame-based specification for representing knowledge. Increasingly, researchers have begun adopting DAML+OIL—itself derived from RDF—as their ontological specification. We use DAML+OIL as SOTA’s descriptive representation language.

Semantic Web, DAML-S, and agents

The need for automatic interoperation of Web services increases as business-to-business and e-commerce applications mature and become more versatile. Unfortunately, developers have designed most of today’s Web content primarily for human interpretation and use. The semantic Web services initiative, which combines the semantic Web and Web services, seeks to make Web services interpretable, thus enabling a wide range of agent technologies that can automatically discover, compose, and execute services.

DAML-S, a semantic markup language based on the DAML language family, has been introduced to represent Web services. It encodes services’ properties, capabilities, interfaces, and effects in concept classes and subclasses, based on an ontology. These semantic markups of multiple Web services collectively constitute a knowledge base, which an agent can populate and query as best suits its purpose.

Agents, acting as conceptual software systems that interact with a virtual environment and exhibit some autonomy, also play an important role in the semantic Web. OntoAgent uses standard markup languages to specify agent behavior and perform tasks actively. It adopts a rule-based approach to handling incoming conditions and applies this approach to automatic document retrieval applications.

SOTA SOLUTION

Using Web services, applications based on various programming languages, object models, and platforms can efficiently and elegantly interact with one another. However, this technology still requires that programmers identify the correct service providers, write binding codes according to users’ needs, test system functionalities, and publish the services so that consumers can find and use them.

Ontology engineering models specific domain knowledge so that machines can better understand the domain’s information. Developers can more effectively automate this work—which requires the sort of comprehension and complex decision making traditionally performed manually—after they prepare the correct model information and make it easily available. A mediating ontology can increase semantic comprehension among different groups of people and systems.

If systems can learn the semantics of using an application programming interface and then integrate the necessary services to automate tasks, users can then tailor how they use information services to fit their own preferences instead of the system designers’.

We designed SOTA to meet this goal. Our work focuses on integrating intranet enterprise applications by leveraging existing legacy systems to provide companies with instant revenue.

SOTA system architecture

The SOTA platform uses a mediating ontology to integrate intranet applications, providing a single integrated user interface instead of separate operations. By modeling the semantic relationships between Web services interfaces and the mediating ontology, SOTA assumes most of the complex tasks previously performed by hand.

Figure 1 presents the overall SOTA platform architecture. During the design phase, SOTA creates a wrapper for existing back-end legacy systems and makes them accessible through Web services interfaces. The domain ontology, which reflects abstract concepts and relationships in the real application domain, provides the architecture’s pivotal element.

When deploying these systems on the SOTA platform, registering the services interfaces’ semantics to corresponding concept properties captured in the mediating ontology is vital. SOTA supports two registering tools for the system annotator—WSDL Semantic Annotator and Ontology Locator. The Task Process Composer tool helps with constructing a reusable task flow for complex tasks that
involve multiple services through the Task Flow Engine.

At runtime, SOTA can take plain-text sentences as inputs and serve end users with a single, integrated user-interface form, avoiding the need to rely on the user’s knowledge and memory to access the necessary distributed services manually. An authorized context-and-content parser deals with different functions and presentation devices in full view of users.

We have found that most daily office tasks SOTA performs have three phases:

- **Searching.** SOTA first parses users’ intentions using four levels of matching, then pushes possible services according to the ratings obtained using the ontology.
- **Engaging.** Next, SOTA prepares most of the input data fields the executing services require, then fetches any semantic relationship retrievable among the operations. The Data Agent helps users analyze and collect all correct and available field data.
- **Processing.** SOTA then combines several individual but sequentially related services as a reusable process, according to specific task requirements and users’ experiences. The Task Agent helps users log all execution paths and thus determines the right task process to invoke multiple services.

**Services semantic registry and ontology design**

When searching for the right Web services, consumers must use a UDDI server Web site or client software developers kit to perform a lookup of readable provider descriptions. Thus, SOTA includes an ontology-based description framework that describes Web services and increases its degree of automation. Unlike the DAML-S approach of providing a detailed semantic description for each service available on the Internet, SOTA uses a centralized ontology and semantic registry.

Analyzing a WSDL document’s structure reveals that each Web service can be understood as follows: Output parameters ← Operation method (Input parameters). When developers register the semantics of each service with the SOTA platform, they must associate each method and its I/O parameters with the corresponding concept or property in the shared ontology.

Web services interfaces that share similar or even identical methods and I/O parameters have often been thought to hold different meanings because of variant naming conventions and expressions. Thus, SOTA introduces two ontology types to achieve a common understanding of heterogeneous Web services. The Operation Ontology models major action types and the Resource Ontology supplies shared meanings for I/O parameters. SOTA currently uses DAML+OIL as the ontology specification language.

**Web services operation ontology.** Here, operation refers to the method name the service interface employs. The operation ontology helps determine the type of operation each Web service performs, and it includes the service’s significant action semantics, primarily through verb classification. For example, a Web service for booking a meeting room, queryAvailableMeetingRoom, uses the QUERY operational concept, which is a surrogate for query, search, list, or similar verbs modeled in the operation ontology.

In our experience, generic database operations such as INSERT, DELETE, UPDATE, and QUERY provide good candidate action concepts and serve as a key component in the operation ontology. When constructing the ontology, we found it best to keep the design simple—complex approaches can make discovery of Web services inefficient. Fragmentary but essential, the operation ontology includes action concepts, their verb synonyms, and an as-is relation among concepts.

**Resource ontology.** This feature facilitates determination of the I/O semantics for a Web service interface. The resource ontology incorporates con-

![Figure 1. SOTA system architecture. A wrapper encapsulates the back-end legacy systems and makes them accessible through Web services interfaces, while the domain ontology provides the architecture's pivotal element.](image)
cepts, noun synonyms, properties, and as-is and part-whole relations among concepts. To ensure the resource ontology’s reusability, SOTA divides it into two categories:

- domain-independent, shared across application domains; and
- domain-dependent, shared within a particular application domain.

Regardless of the application domain to which it is applied, a domain-independent ontology includes generic concepts such as TIME and USER. Figure 2 shows an example of a Web service method, Oper2: bookMeetingRoom, which requires four input parameters—userID, startDate, endDate and meetingRoom—and generates a result represented by a single output parameter, recordId, described in WSDL.

The corresponding resource ontology support works as follows. The userID parameter is a User_id property of the BOOKING_RECORD concept in a domain-dependent ontology—in this instance, the office domain. This parameter is also the Id property of the USER concept in the domain-independent ontology. Similarly, the startDate parameter is the Start_date property of the BOOKING_RECORD concept, whereas the Start_date property lies semantically within the START_DATE range.

Semantic registry of the WSDL interface. For conveniently registering Web services, SOTA includes a WSDL semantic annotator tool to parse a WSDL document and capture its methods and I/O parameters, based on operations and resource ontology. SOTA then stores the Web services semantics in a Web service semantic registry (WSSR). Additionally, an ontology locator tool efficiently searches and browses the ontology in a star view to find appropriate semantics.

Figure 3 shows an instance of using both tools to register a service for sending e-mail. Once the WSDL file loads the operation name sendMail, the WSDL Semantic Annotator extracts the output parameter Transaction_Status and input parameters such as Email_Title and EMailReceiver. The semantic annotator and ontology locator tools can then provide the service’s semantics.

When working on an operation, the developer can input a plain-text description and operation concept—SEND, in this case. When working on a parameter, the developer can enable any of three properties: Editable means the parameter can accept arbitrary user input, Recordable means the parameter’s value is worth memorizing for subsequent use, and Multiple means the parameter accepts multiple values such as multiple e-mail recipients. To identify which concept or property each parameter should link to, the developer can locate the appropriate entity by searching or browsing the premodeled ontology via the Ontology Locator. In this case, SOTA relates the parameter string_1 to an Email_Title property of the ontology’s Email concept.
Parsing requirements and matching services
SOTA provides an effective Web service lookup engine. Figure 4 shows how the WSLE operates. It first identifies ontological meanings and their particular values from a sentence the user inputs to specify a semantic pool of intention.

The WSLE applies four types of intention identifications. Syntax extraction identifies predefined noun phrase patterns such as time expression, place format, or employee role. Operation identification obtains the action concept by verb classification in the Operation Ontology. Resource identification determines the noun concept or property that matches the DAMLClass. Instance identification determines the instance or individual that matches the DAMLInstance modeled in the resource ontology.

For example, a user could input “I want to get Doctor Yang’s schedule for Dec 9.” The WSLE identifies

- “Dec 9” as a value of the Date_of_the_TIME concept;
- “get” as a synonym of the SEARCH concept anchored in the operation ontology;
- “schedule” as a synonym of the SCHEDULE, PROJECT_SCHEDULE or PERSONNEL_SCHEDULE concept; and
- “Doctor Yang” as an instance of CHIEF under the EMPLOYEE concept.

Using this process, the WSLE generates the intention semantic pool {Date_of_the_TIME, SEARCH, SCHEDULE, PROJECT_SCHEDULE, PERSONNEL_SCHEDULE, CHIEF_SCHEDULE}.

Completing the service form
Office workers often must run various programs and complete forms in many applications to accomplish their daily tasks. The pitfalls commonly encountered when completing these forms include redundancy of data and a failure to integrate the data that various programs generate. Users must repeatedly input user-specific data manually and memorize the key data that link different programs’ functions. They also must look up external information resources for data not related directly to the task.

Working with the service registry mechanism, the SOTA framework presents a single form for all matched services. The Data Agent helps gather all necessary contextual information, including duplicate, similar, and relevant types, then feeds this data into the form automatically.

The form usage pattern reveals three data types that users commonly enter into a form. Historical data includes any data a user enters into a form.
Session data refers to all data input and output during a user’s current session. Environmental data refers to resources that may be physically observed or virtually modeled in a user’s working environment. SOTA includes a form-based and session-based memory mechanism based on these classifications that lets the Data Agent collect the data that users seek. SOTA’s data acquisition environment (DAE) provides the architectural basis for session-based memory. Figure 5 shows the overall form-completion operation.

**Form memory—persistence of form data.** The fundamental issue in dealing with historical data is ensuring the persistence of meaningful information. Only some data is worth preserving. For example, a user’s address is meaningful, whereas his appointment date with his dentist two weeks ago is not. Also, data access time and frequency reflect users’ expectations. Thus, the form memory should present data that historically has had the highest hit rate.

**Data acquisition environment.** Say, for example, a user wants to find the current project leader’s name. SOTA uses DAML+OIL and the DAE to answer this query by modeling the concept of a user with a DAMLClass that has a DAMLProperty, name, with a literal range and another DAMLProperty, project_leader, which describes the relationship between two user concepts. The project_leader relationship can be formed if a representative service exists to identify which user instance is the project leader of the current user instance, which is why we use the DAE to invoke this representative service. The DAE thus can be viewed as a container of concept instances, each of which can have a specific property that binds to other existing instances. By investigating the underlying ontological structure, SOTA can create a virtual data path in the DAE for every pair of loosely related concept instances.

SOTA can determine the relationship between two concept instances or it can bind them statically or dynamically. For example, the data agent can look up the service registry to perform late binding dynamically through a series of service invocations. An example would be to query the employee ID of the current project leader, then to query the name of the employee with that ID. The Data Agent aggregates related data of the concept instance—either literal values or instance values—accordingly by invoking services. SOTA constructs many data paths in the DAE. The Data Agent can use these paths to determine which data to feed into the form field. Together, the DAE and the Data Agent provide semantic understanding and reasoning at the instance level, which lets SOTA consider the resulting data paths as data-locating anchors in the DAE.

**Session memory.** Based on the infrastructure the DAE provides, a session memory—basically, a DAE with a Data Agent—stores and fetches data of existing instances. The Data Agent records and stores all of a phase’s manual user input data and user-session service-execution results as a session object. The Data Agent uses this session object to collect the preferred data about a specific or general concept instance, according to the semantic knowledge in the service’s form field. The Data Agent can fetch the environment data as a concept instance whenever required. The DAE
will look up the service registry and invoke the service if the agent defines and tags the concept as a retrievable instance.

**Modeling and executing task flow**

SOTA forms each office environment task flow to complete a specific task. It can form a new task flow automatically or the Task Flow Composer visualization tool can compose it manually. The Task Agent memorizes every sequence of services each user executes and logs every data transmission associated with each service. When the Task Agent detects a user performing a service execution sequence frequently, it recommends that the current user forge the execution sequence as a new task flow. In this case, the Task Agent will ask the user to verify the linkage mapping of each task flow parameter.

Figure 6 shows the new Task Flow Composer user interface. When the confidence associated with forming a new process exceeds a specified threshold—0.9 in this instance—the system prompts the user to confirm formation of a new task flow. The Task Flow Composer lists all the flow’s services and service I/O parameters and associates each parameter with a semantic linkage or preferred data instance. The semantic link helps combine semantically related I/O parameters to eliminate duplicate inputs at runtime. If any service in the flow causes an exception, the exception handler searches for the exception-handling service before it continues with the next step. The parameters that govern this service derive from the task flow’s current state, according to the data’s conceptual mapping.

**APPLICATION SCENARIO**

We have entered into the SOTA WSSR registry all the office services and regular task flows in our laboratory along with their interfaces and their semantic meanings. The platform records all task procedures and data transactions involved in the employees’ performance of their daily tasks.

In a typical scenario exploring SOTA’s technical feasibility, Eric, an office worker who is currently away from his desk, wants to arrange a meeting with his boss. In the past, performing this task required entering 36 parameters. With SOTA, Eric now fills in only four indispensable data fields.

When Eric types “Meeting with boss” into his PDA, SOTA turns this phrase into a task requirement. In response, the WSLE extracts the task inten-
tion from the requirement and puts it into the intention semantic pool, from which the Task Agent identifies an appropriate task flow. In this case, the task flow consists of the four activities shown in Figure 7.

Each activity initiates a Web service operation for the corresponding task procedure. First, the Data Agent looks up environment data such as meeting rooms and projectors from the current session memory. It then selects appropriate data values based on the preset preferences.

Next, Data Agent fetches various meeting text fields—Title, Content, Memo, Description—from the persistent form memory. Using a data path, such as [Employee→(belong to)→Project→(has members)→Employees], SOTA determines this meeting’s participants by binding the current user instance, Eric, to member instances of Eric’s team: Han-Kuan, Brian, and Snowball.

Having collected sufficient information to perform the task, Data Agent feeds the collected data into the task flow and passes control to Task Agent, which carries out the task flow and handles exceptions according to user preferences. For example, if meeting room B is unavailable, Task Agent automatically looks for the preferred alternative solution, meeting room A. Task Agent reports the final meeting arrangement to Eric for confirmation and subsequent execution.

A conventional application-centric platform requires significant user effort to find systems and log onto them, select functions using the mouse, and copy and paste the data. In contrast, SOTA is a user- and task-centric platform that shifts effort away from users so that they can complete their tasks more efficiently.

Enabling remote invocation among heterogeneous systems and hence delivering a solution that leverages diverse legacy systems requires more work in the Web services field.

To address the security issues that are always a concern, even in an intranet environment, work to establish a common authorization service is under way, and a solution is expected soon. Work in process-behavior pattern analysis is focusing on improving current interactive models to combine them with workflow systems. Beyond using data and task agents, a context-aware user agent can contribute to improved personalization and adaptivity.

To facilitate automation of modeling the ontology, we are investigating the use of an incremental approach that can add the power of self-learning and self-adaptation to the execution and approval of solution, process, service, or parameter mapping.

Acknowledgments

This study was partially supported by the Center for Information and Electronics Technologies, National Taiwan University, and by the Institute for Information Industry’s Innovative and Prospective Technologies Project, which is sponsored by the Minister of Economic Affairs, Republic of China. We thank Kwei-Jay Lin and the anonymous reviewers for their insightful and valuable comments. Portions of this article were presented at the 2003 International Conference on Database and Expert Systems Applications.¹
References


Tse-Ming Tsai is an engineer at the Institute for Information Industry’s Advanced e-Commerce Technology Laboratory, Taipei, Taiwan. His research interests include Web computing, software engineering, and e-commerce. Tsai received an MS in information management from National Taiwan University. Contact him at eric@iii.org.tw.

Han-Kuan Yu is an engineer at the Advanced e-Commerce Technology Laboratory. His research interests include ontology engineering and the semantic Web. Yu received an MS in information management from Ming Chuan University. Contact him at bankuan@iii.org.tw.

Hsin-Te Shih is an engineer at the Advanced e-Commerce Technology Laboratory. His research interests include ontology engineering and information retrieval. Shih received an MS in management information science from National Yunlin University of Science & Technology. Contact him at snowball@top-team.com.tw.

Ping-Yao Liao is an engineer at the Advanced e-Commerce Technology Laboratory. His research interests include digital signal processing and object-oriented programming. Liao received an MS in electronic engineering from National Tsing Hua University. Contact him at brian@iii.org.tw.

Ren-Dar Yang is a vice general director at the Advanced e-Commerce Institute. His research interests include Web services, component-based software development, and knowledge management. Yang received a PhD in computer science and information engineering from National Chiao-Tung University. Contact him at rdyang@iii.org.tw.

Seng-cho T. Chou is a professor in the Department of Information Management at National Taiwan University. His research interests include Web-based information systems, knowledge management, and e-commerce. Chou received a PhD in computer science from the University of Illinois at Urbana-Champaign. Contact him at chou@ntu.edu.tw.

Looking for a community targeted to your area of expertise? IEEE Computer Society Technical Committees explore a variety of computing niches and provide forums for dialogue among peers. These groups influence our standards development and offer leading conferences in their fields.

Join a community that targets your discipline.

computer.org/TCsignup/