Bringing Web Principles to Services: Ontology-Based Web Services

Muhammed J. Al-Muhammed,[1,*] David W. Embley,[1,*] Stephen W. Liddle,[2,†]
and Yuri A. Tijerino[3,‡]
[1]Department of Computer Science
[2]Information Systems Department
Brigham Young University, Provo, Utah 84602, U.S.A.
[3]Department of Applied Informatics, Web Science Lab
Kwansei Gakuin University, Sanda, Hyogo 699-1337, Japan

Abstract

Researchers are beginning to realize the potential of web services that can use the web as a place for information publication and access as opposed to the traditional web-services paradigm that merely uses the web as a transport medium. Traditional web services can be difficult to discover, can have complex invocation APIs, and require strong coupling between communicating applications. In previous work, we presented ontology-based techniques in which users make service requests using free-form, natural-language-like specifications. This paper shows how we can use these ontological techniques to automatically create ontology-based web services that (1) are easy for software agents to discover because they are created based on machine-processable formalisms (ontologies), (2) have invocation APIs requiring only simple read and write operations, and (3) require no a priori agreements regarding types and data formats between communicating applications. Experiments with our prototype implementation in several domains show that our approach can effectively create web services with these characteristics.

1. Introduction

Despite the tremendous success of traditional web services, researchers have recently realized that web services can be even more successful if they are based on the more fundamental resource publication and access principles of the web [6, 9]. To use a traditional web service, a service requester must discover a service of interest and synchronously communicate with it. A requester must also comply with service-specified data formats and data-exchange protocols. In contrast, *web-principled services* (those fundamentally based on web principles, which we describe in this paper) have none of these requirements.

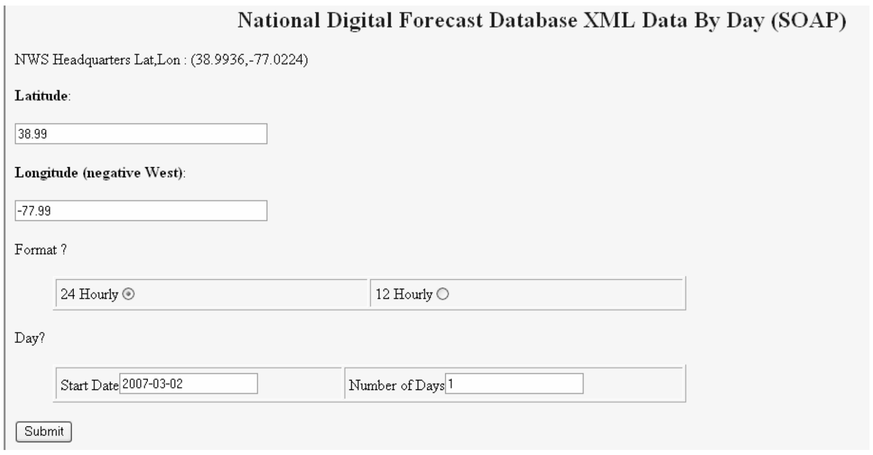
Figure 1 shows a traditional web service that returns a weather report for a given place and time. Assuming a service requester has located this service, the requester would provide a latitude, a longitude, and a date, and may choose the number of days for the forecast and whether the results should be returned in 12-hour or 24-hour increments. The latitude and longitude must be real numbers, not, say, degrees and minutes. Also, rather than specifying "East" or "West", negative values indicate West longitudes. The date must be in the format *yyyy-mm-dd*, not in any of many other possible formats. Further, after clicking on "Submit", the calling application must stay coupled with the service until it returns a response. Figure 2, on the other hand, shows the interface for a prototype for a web-principled service. Without needing to discover a service, a user posts a request. In our example, the requester enters, "What is the weather going to be like tomorrow in Springfield, Illinois?" A service that can appropriately respond observes the posting and responds. The response is twofold: (1) it highlights the part of the query it "understands" (*weather*, *tomorrow*, and *Springfield, Illinois* in Figure 2) and (2) it answers the query (*MaximumTemperature*=70, *MinimumTemperature*=38, and *PercentChanceOfPrecipitation*=10 in Figure 2).

Web-principled services allow decoupling among communicating applications, but, as a result, require heterogeneity resolution.

*Supported in part by the National Science Foundation under Grants 0083127 and 0414644.

†Supported in part by the Rollins Center for eBusiness at BYU under Grant EB-05046 and by the National Science Foundation under Grant 0083127.

‡Supported in part by the National Science Foundation under Grant 0414644.



National Digital Forecast Database XML Data By Day (SOAP)

NWS Headquarters Lat,Lon : (38.9936,-77.0224)

Latitude: 38.99

Longitude (negative West): -77.99

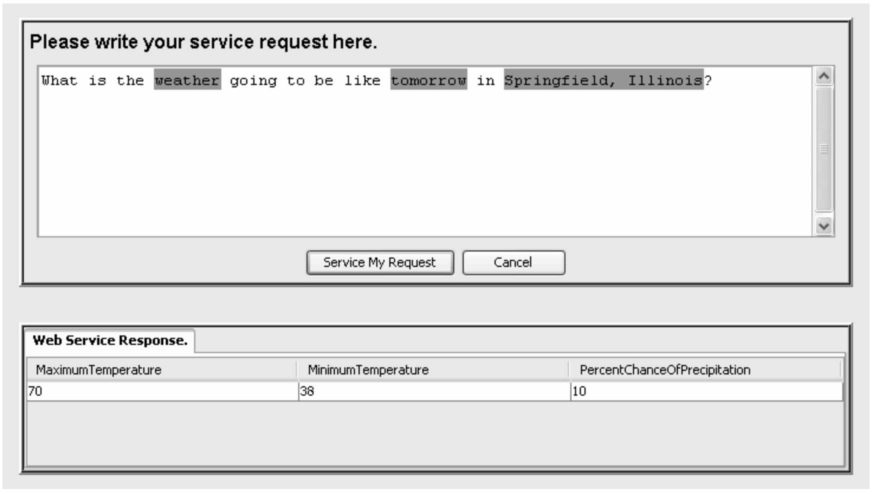
Format ?

24 Hourly [✓ ●] 12 Hourly ○

Day?

Start Date 2007-03-02 Number of Days 1

Figure 1. A weather report web service.



Please write your service request here.

```
What is the weather going to be like tomorrow in Springfield, Illinois?
```

Web Service Response.

MaximumTemperature	MinimumTemperature	PercentChanceOfPrecipitation
70	38	10

Figure 2. A free-form weather report request with all recognized constraints and values highlighted along with the web service response to the request.

- *Decoupling among communicating applications*. In our vision of web-principled services, service requesters post their requests to the web without the requesters having to reference any particular service. Requesters, therefore, do not need to discover these services nor to communicate directly with them. As a result, a decoupling of services and requesters is achieved. To negate the need for service discovery and direct communication, however, web-principled services must be able to read posted service requests, process them, and return results to requesters.

- *Heterogeneity resolution*. For successful invocation of a service, posted service requests must comply with the service-specified data. Prior agreement between a requester and a service on the data formats, types, and mappings between values in a request and the respective input parameters of a service would enable compliance. Decoupling, however, prevents prior agreement. Therefore, web-principled services must make requests comply with their internal data specification by resolving heterogeneity.

In previous work [3, 1, 2], we have proposed an ontological technique that allows service requesters to invoke certain types of ontology-based services using free-form, natural-language-like specifications. We explain in this paper how these ontology-based services satisfy the decoupling and heterogeneity requirements and thus can enable web-principled services. We call these services *ontology-based web services* (*OBWSs*). OBWSs interact asynchronously with service requesters. OBWS invocation only re-

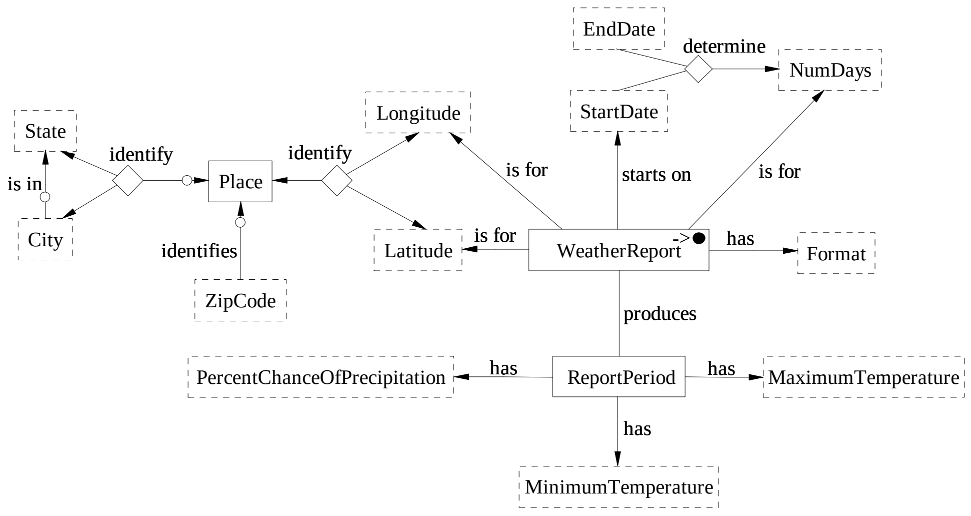


Figure 3. A semantic data model for a weather report service.

quires reading free-form service requests from some resource such as the web. Thus they satisfy the decoupling requirement because they do not need to have a direct coupling with the service requester. OBWSs require no prior agreement on exchanged data. An OBWS recognizes values in a service request such as the highlighted values in Figure 2 and binds them to its input variables. In order for this binding to be successful, the OBWS (1) casts each recognized value to a type of an input variable, (2) transforms the format of each recognized value to an internal format conforming to the format specified by its ontology, and (3) assigns each recognized value to the respective input variable. In addition, an OBWS identifies any missing values necessary for the invocation and obtains them, seeking additional information from the requester, if necessary. All these capabilities of an OBWS in handling a service request enable it to satisfy the heterogeneity requirement.

We give the details of our contributions to enabling web-principled services via OBWSs as follows. Section 2 introduces ontology-based web services. Section 3 presents the OBWS architecture and shows how it meets the decoupling requirement for web-principled services. Section 4 explains how our approach can bring web principles to traditional web services, emphasizing how our techniques provide a way for invoking traditional web services. Section 5 compares our approach to related work. In Section 6, we give concluding remarks and directions for future work.

2. Ontology-Based Web Services

We now discuss the elemental components of ontology-based web services. Section 2.1 introduces the semantic data model that we use to represent ontologies, and Section 2.2 introduces how we capture the semantics of the instances of different concepts of the semantic data model. Section 2.3 shows how OBWSs service requests and how they meet the heterogeneity resolution requirement.

2.1. Semantic Data Model

A *semantic data model* specifies named sets of objects, which we call *object sets*, named sets of relationships among object sets, which we call *relationship sets*, and constraints over object and relationship sets. Figure 3 shows a domain ontology for providing a weather report. The domain ontology consists of object-set concepts such as *StartDate* and *Place* that can be used to make requests for weather reports. The semantic data model has two types of object sets, those that are lexical (enclosed in dashed rectangles) and those that are nonlexical (enclosed in solid rectangles). An object set is *lexical* if its instances are indistinguishable from their representations. *StartDate* is an example of a lexical object set because its instances (e.g. "May 7, 2007") represent themselves. An object set is *nonlexical* if its instances are object identifiers, which represent real-world objects. *Place* is an example of a nonlexical object set because its instances are identifiers such as, say, "P_1", which represents a particular place in the real world.

We designate the main object set in a domain ontology by marking it with "–> •" in the upper right corner (e.g. *WeatherReport* in Figure 3). This notation,"–> •", denotes that when an ontology is used to satisfy a service request, the main object set becomes ("->") an object ("•"). The system satisfies a service request by instantiating the main object set with a single value such that all applicable constraints are satisfied.

Figure 3 also shows relationship sets among object sets, represented by connecting lines, such as *StartDate and EndDate determine NumDays*. The arrow connections represent functional relationship sets, from domain to range, and non-arrow connections represent many-many relationship sets. For example, *StartDate and EndDate determine NumDays* is functional from the pair *StartDate* and *EndDate* to *NumDays*, and *WeatherReport produces ReportPeriod* is many-many. A small circle near the connection between an object set *O* and a relationship set *R* represents an optional participation, so that an instance of *O* need not participate in a relationship in *R*. For example, the small circle on the *Place* side of the relationship set *ZipCode identifies Place* states that an instance of *Place* may or may not relate to an instance of *ZipCode*.

2.2. Data Frames

Each object set in a semantic data model has an associated data frame [5], which describes instances for the object set. Data frames capture the information about object-set instances in terms of internal and external representations, context keywords or phrases that may indicate their presence, operations that convert between internal and external representations, and other manipulation operations that can apply to instances of the object set along with context keywords or phrases that indicate the applicability of an operation and operands in an operation. Figure 4 shows sample (partial) data frames for several object sets in Figure 3.

The internal representation specifies the data type for the instances of an object set. The *StartDate* instances, for example, are of type *date*, which must be of the form *yyyy-mm-dd*. We use regular expressions to capture external textual representations. The *StartDate* data frame, for example, captures date instances such as "July 6, 2007". The regular expression can also have lexicon references. For instance, "{stateName}" in the *State* data frame refers to a lexicon of state names. A data frame's context keywords/phrases are also regular expressions. For example, the *ZipCode* data frame in Figure 4 includes context keywords such as "zip code" or "zip". In the context of one of these keywords, if a 5-digit number appears, it is likely that this number is a zip code. A nonlexical object set such as *WeatherReport* has only context keywords or phrases. Figure 4 shows that the *WeatherReport* data frame includes keywords and phrases that could indicate the presence of an instance of a *WeatherReport*.

Operations in data frames manipulate object-set instances. For example, the operation *getLatitude*(*x*: *ZipCode*) computes the latitude for a given zip-code argument *x*, and the operation *toInternalRepresentation*(*x*: *StartDate*) transforms dates in various formats to the internal format. Context keywords/phrases for an operation indicate the possible applicability of the operation. Context keywords/phrases are regular expressions that include keywords or phrases and possibly expandable expressions represented by operand names enclosed in braces. When context keywords/phrases for an operation match substrings in a service request, the system can record which values are for which operands. For instance, the context keywords/phrases associated with the operation *NrDaysBetween* in Figure 4 has the regular expression *between*\s+{*x1*}\s+*and*\s+{*x2*}, which includes the expandable expressions {*x1*} and {*x2*}. As Figure 4 shows, the operands of these two expressions are of type *StartDate*. When this regular expression matches a substring in a request such as "What is the weather going to be in Chicago between the 10th and the 15th," the system can record that the first date value ("the 10th") is for *x1* and the second date value ("the 15th") is for *x2*.

2.3. Servicing Requests with Ontology-Based Web Services

When an OBWS receives a service request, it applies the recognizers in the data frames of its underlying ontology to the request to extract information necessary for servicing the request. Consider the weather-report request in Figure 2 and an OBWS whose underlying ontology is in Figures 3 and 4. When applying the ontology in Figures 3 and 4 to the request in Figure 2, the OBWS extracts the strings "weather", "tomorrow", "Springfield", and "Illinois"—these strings appear highlighted in Figure 2.

The OBWS may, and often does, need to process the extracted information to make it comply with the required data as specified by the ontology. In our weather-request example, some of the extracted values do not comply with the data required by the OBWS. In this case, the request provides a state ("Illinois") and a city ("Springfield"), not a latitude and longitude as required. The OBWS uses the operation *getLatitude*("*Illinois*","*Springfield*") to compute a latitude and assigns the computed latitude value to the object set *Latitude* (similarly for a longitude). Likewise, the request provides "tomorrow" as a start date, which is not in the required format *yyyy-mm-dd*. Since the applicability recognizers of the operation *Tomorrow*() in the data frame recognize "tomorrow" in the request, and this operation returns a start date, the OBWS uses *Tomorrow*() to transform the recognized string "tomorrow" into a properly formatted start date and assigns the transformed value to the object set *StartDate*. Besides not being in the proper form, the information in the request also is incomplete since the request provides no values for *Format* and *NumDays*. The OBWS, nevertheless, can use its ontology to provide default values for object sets.[1] For our example the OBWS provides

[1]When there are no default values for required information, the ontol-

```
StartDate
  internal representation: date -- format: yyyy-mm-dd
  text representation: {monthName}\s+([0]?[1-9]|[12]\d|3[01])(\s*\,)?\s+\d{4}|
                       (the\s+)?([0]?[1-9]|[12]\d|3[01])\s*(th|nd|rd|st)|...
  toInternalRepresentation(x: StartDate) returns (StartDate) -- format: yyyy-mm-dd
  Tomorrow() returns (StartDate) -- next day with respect to today
    context keywords/phrases: tomorrow|next\s*day|...
  NrDaysBetween(x1: StartDate, x2: EndDate) returns (NumDays)
    context keywords/phrases: between\s+{x1}\s+and\s+{x2}|...
  getDefaultStartDate() returns (StartDate) -- today's date
  ...
State
  internal representation: string
  text representation: {stateName}|{statePostalCode}|{stateAbbreviations}
  ...
ZipCode
  ...
  text representation: [1-9]\d{4}
  context keywords/phrase: zip\s*code|zip
Latitude
  internal representation: real -- positive
  getLatitude(x1: State, x2: City) returns (Latitude)
  getLatitude(x: ZipCode) returns (Latitude)
  toInternalRepresentation(x: Latitude) returns (Latitude) -- positive real number
  ...
Longitude
  internal representation: real -- negative
  getLongitude(x1: State, x2: City) returns (Longitude)
  ...
NumDays
  internal representation: integer
  getDefaultNumDays() returns (NumDays) -- 1
  ...
WeatherReport
  internal representation: object ID
  context keywords/phrases: (want\s+a\s+)?weather\s+report|weather|forecast|...
  ...
```

Figure 4. Some sample data frames (partial).

the default value "24 Hourly" for *Format* and the default value"1" for *NumDays*. Thus, the OBWS can instantiate all the required data *Latitude*, *Longitude*, *StartDate*, *NumDays*, and *Format*, either by computing or transforming, if necessary, given values or by providing default values.

To service a request, an OBWS must instantiate its output data instances by retriving them from its database or by computing them from other available data instances. For our example, the OBWS retrieves from its database values for *MaximumTemperature*, *MinimumTemperature*, and *PercentChanceOfPrecipitation*. Using a process that is beyond the scope of this paper and is fully described elsewhere [3], the OBWS generates a database query, executes this query, and returns the query result values.

Observe that OBWSs resolve data heterogeneity. An OBWS extracts values from a request and assigns them to respective object sets of its ontology independently of how these values appear in the request. An OBWS uses the data frames operations to compute required values from given values and to transform recognized values to internal representations when necessary. An OBWS also uses default values defined by its ontology to add missing information to service requests.

ogy lets the system know precisely which values are needed. The OBWS can request these values from the user.

3. Request-Oriented Architecture

Fensel [6] proposes triple-spaced computing as a means to provide seamless interoperability between web services in a web-principled manner. In this section we introduce a *request-oriented architecture*, inspired by the feed syndication architecture of XML- or RDF-based RSS and ATOM feeds, that meets these additional interoperability requirements. This architecture provides a mechanism to *dynamically* match service requests with OBWSs capable of servicing those requests. Fundamentally, this architecture is based on a centralized brokerage mechanism that enables OBWSs to subscribe to service-specific ontology feeds to which new service requests are posted. The brokerage mechanism's main functions are to match requests against available ontologies and to update service-specific ontology feeds with service requests so that OBWSs that subscribe to the feed can receive and service relevant service requests.

In the request-oriented architecture, service requesters make free-form service requests such as the weather request in Figure 2. The broker matches a request with the available ontologies. For each ontology, the broker applies all the recognizers in the data frames to the request and ranks the ontology according to the number of matches the ontology has with the request. (More details about request-ontology matching and ranking can be found elsewhere [2].) The broker then selects the ontologies with the highest rank as a means to find OBWSs that can service the request.

The service brokerage mechanism takes a two-tiered approach to broker requests and responses between services requesters and OBWSs. The first tier consists of mapping a service requester with potential OBWSs associated with the ontology that the broker determined to be most relevant to the service request. In this tier, the request is published as an update to the ontology feed. Then the OBWSs that subscribe to the feed may choose to respond to the broker based on their availability and other individual criteria. The brokerage mechanism then presents the OBWS response proposals to the service requester as links. In the second tier of brokerage, the service requester selects a response proposal from the list provided by the brokerage mechanism which indicates to the brokerage mechanism the acceptance of the proposed service response. At this point, the brokerage mechanism performs the last step of the brokerage process by establishing a direct link between the service requester and the selected OBWS. This step consists of sending the complete service request to the selected OBWS along with the service requester's URL, the web session ID, and other relevant parameters in the POST request necessary for the OBWS to interact directly with the service requester. After the direct link between the service requester and the OBWS is successfully established, the OBWS then provides the complete response to the service requester and opens the door for further direct interactions.

Observe that the proposed OBWS architecture decouples service requesters and OBWSs. Requesters do not have to discover OBWSs that are capable of servicing their requests; the broker selects the capable OBWSs via request-ontology matching. In addition, requesters do not have to reference and establish communication links with the OBWSs; the broker references and dynamically establishes the communication links.

The broker scales up to a reasonable size. The average size of the ontologies with which we have been experimenting is 18KB. Therefore, for 10,000 ontologies, the required space is about 180MB, which is relatively small for recent machines. Based on [12], for *r* regular expressions with average length of $\bar{n}$ characters, a text of length *t* characters, a machine of a clock speed s, the time to process the text using the regular expressions is $r\bar{n}t/s$. Since our ontologies have an average of 90 regular expressions with an average of 60 characters, processing a request of 70 characters, like the one in Figure 2, on a machine with a clock speed of 1.8GH would take about 0.20 milliseconds. Thus, for 10,000 ontologies the time is about 2 seconds. We add to this the time for applying lexicons, which each can be done in (log_2L) time, where L is the length of the lexicon. In any case, the time complexity is still manageable and can be greatly improved by using techniques such as parallel processing, duplicate regular-expression removal, and indexing.

4. Web-Principled Traditional Web Services

This section describes how to turn a traditional web service into an OBWS, not only so that it can be invoked via the OBWS architecture, but, perhaps more importantly, so that it can also exploit the decoupling and data heterogeneity resolution capabilities of the OBWS approach.

To create an OBWS from a traditional web service, we must provide an ontology that describes the service, and we must specify mappings between the ontology and the service I/O requirements. A developer can reuse an ontology, if there is one, or create a new ontology. No change to the actual interface and the internal implementation of the service is required. The information required to create an ontological description for a traditional web service includes (1) the name of the service, (2) the names of the input and output parameters along with their types, and (3) the accepted formats for the values of the input parameters. Figures 3 and 4 show the ontology that describes the weather web service example in Figure 1.The semantic model in Figure 3 encodes the input parameters of the service in terms of the object sets *Latitude*, *Longitude*, *StartDate*, *NumDays*, and *Format*. The semantic model also encodes the output parameters of the service in terms of the object sets *MaximumTemperature*, *MinimumTemperature*, and *PercentChanceOfPrecipitation*. We define the name of the operation as the main object set (marked with "–>•") in Figure 3.

The data frames in Figure 4 define the semantics of the possible instances of the object sets. The types of the object sets must comply with the types of the input and output parameters required by the service. As an example, the WSDL document that describes the interface for the weather service in Figure 1 defines the input parameter `startDate` to be of type `date` (as defined in XML Schema). Therefore, Figure 4 defines the type of the object set *StartDate* to be of type `date`.

The ontology creator may also define extensions to the input of the original service in Figure 1. It declares additional object sets *State*, *City*, and *ZipCode*. With these

```
WeatherRequest(
 getLatitude("Illinois","Springfield"),
 getLongitude("Illinois","Springfield"),
 Tomorrow(),
 getDefaultNumDays(),
 getDefaultFormat()
 )
```

(a) An instantiated service request.

```
<?xml version="1.0" ?>
 <dwml ...>
 ...
  <temperature ...>
   <name>Maximum Temperature</name>
   <value>70</value>
  </temperature>
  <temperature ...>
   <name>Minimum Temperature</name>
   <value>38</value>
  </temperature>
  <probability-of-precipitation ...>
    <name>Probability of
       Precipitation</name>
    <value>10</value>
  </probability-of-precipitation>
 ...
 </dwml>
```

(b) The service response for the request in Figure 5(a) (partial).

Figure 5. An instantiated request and the service's response to this request.

added object sets, a requester can invoke the service in Figure 1 not only using a latitude and a longitude, but also using a zip code, a city, or both a city and a state. It also declares *EndDate*, which provides an alternative way to specify the number of days for the forecast.

To map the request to the input requirements of the service, the developer must specify which input parameter map to which object set of the ontology. In our ontologies we do this mapping through the data frames by making each input parameter name a synonym for the respective object set in the ontology. In this way, the OBWS can pass the internal representation of an extracted value to the respective input parameter. Figure 5(a) shows the results of the mapping for our example. The developer must also map the service response to the ontology. Typically web service responses are XML documents embedded in SOAP messages. The developer therefore must specify which XML tag corresponding to which object set. Then, the developer can use XSLT or XML parsers to extract values and assign these values to appropriate object sets. Figure 5(b) shows an example of the output from the service, which the developer must map to the ontology for our service.

5. Related Work

There are several existing projects related to our work. Some frameworks, such as the Semantic Web Service Framework (SWSF) [13], OWL-S [10], and WSMX [8], can be used to develop web services from scratch by describing their internal representation with semantic data models (e.g. SWSF uses Semantic Web Service Ontology and a Semantic Web Service Language for this purpose). Others, such as the Semantic Annotation for the Web Services Description Language (SAWSDL), provide a means to create an annotation of an existing web service's interface, while not being constrained to any ontology in particular. Perhaps most similar to our approach are those that allow for both internal and external semantic modeling of the web service. These include the Web Service Modeling Framework (WSMF) [7] and IRS-II [11].

WSMF provides decoupling among applications through an ontology modeling language and an ontology. However, this decoupling is partial because mapping between ontologies and the service data needs to be done manually. In contrast, our approach uses data frames to automate the mapping process. In addition, the WSMF is a framework that provides no web service implementation capabilities, while OBWS does.

Approaches [8] and [11] build upon the WSMF by letting service developers describe their services using the Web Service Modeling Ontology (WSMO) and register these descriptions on the server. Requesters can specify their requests, called goals, also using WSMO. The server then matches goals with WSMO service descriptions and returns matches to users, who choose and invoke the desired service. Both of these approaches differ from OBWS in that they do not handle strong data heterogeneity as our approach does. They do, however, resolve mismatches between the request ontology and the service description ontology using prespecfied mapping rules that must be created manually. Further, our approach allows users to make free-form requests rather than using the formal WSMO ontology.

WSMF, IRS-II, and WSMX are service-oriented approaches that partially accomplish *decoupling* and *heterogeneity* requirements through the conceptual modeling capabilities of ontologies and logic-based languages used to either describe the interface or define the inner workings of the web services. The OBWS approach described in this paper, on the other hand, is an request-oriented approach, which allows requests to be profiled with a extensional ontology and posted on an ontology feed, so that services subscribed to the feed can service the request. Neither the requester nor the servicer need to worry about each other's data representation, transport protocols, internal or external representation, or other idiosyncrasies in order to communicate.

6. Conclusions and Future Work

We have presented an ontology-based approach to enable web-principled services via ontology-based web services (OBWSs). Web-principled services use the web as a place for information publication and access. They communicate asynchronously (and thus resolve coupling problems), and they exchange data without requiring requesters to comply with strict data format specifications (and thus resolve heterogeneity problems). In addition, we have proposed an architecture for OBWS. Instead of a passive mechanism such as that embodied in UDDI-based brokerage services that requires the service requester to find, adapt, and request, our proposed mechanism allows requests to be advertised in a manner that an OBWS can understand them and then propose to service those requests. Instead of a reactive mechanism that depends on human developers to make the necessary adjustments to the interacting requester and responder, the proposed mechanism provides proactive mapping between requests and responses through its relevant ontology-based feeds. As a spin-off of the basic OBWS framework, we have also described how to turn a traditional web service into an OBWS. It suffices to describe a traditional web service with an ontology; it requires no changes to the interface and implementation of the service.

There are two important directions left for future work. First,we need to provide a mechanism for the broker to be able to choose the best service for a request when there are many relevant service providers that match the request. Others such as [4] have suggested criteria for selecting a service from among potential services according to non-functional aspects such as reliability, service cost, and availability. We can adapt these criteria to our approach or use any other appropriate techniques. Second, we want to extend our approach to handle composite services whose satisfaction requires an instantiation of multiple main object sets. For instance, a vacation planning web service should book an air ticket, reserve a hotel, and rent a car. Handling this type of a service, however, is not just a matter of independently instantiating the main objects. There are clearly cross constraints that need to be satisfied in order for vacation planning requests to be correctly handled. For instance, the date of the car rental cannot be later than the return date of the air ticket.

References

[1] M. J. Al-Muhammed and D. W. Embley. Resolving Underconstrained and Overconstrained Systems of Conjunctive Constraints for Service Requests. In *Proceedings of the 18th International Conference on Advanced Information Systems Engineering (CAiSE06)*, pages 223–238, Luxembourg, June 2006.

[2] M. J. Al-Muhammed and D. W. Embley. Ontology-Based Constraint Recognition for Free-Form Service Requests. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE 2007)*, pages 366–375, Istanbul, Turkey, April 2007.

[3] M. J. Al-Muhammed, D. W. Embley, and S. W. Liddle. Conceptual Model Based Semantic Web Services. In *Proceedings of the 24th International Conference on Conceptual Modeling (ER 2005)*, pages 288–303, Klagenfurt, Austria, October 2005.

[4] D. Claro, P. Albers, and J. Hao. Selecting Web Services for Optimal Composition. In *Proceedings of the 2nd International Workshop on Semantic and Dynamic Web Processes (SDWP 2005)*, pages 32–44, Orlando, Florida, July 2005.

[5] D. W. Embley. Programming with Data Frames for Everyday Items. In D. Medley and E. Marie, editors, *Proceedings of AFIPS Conference*, pages 301–305, Anheim, California, May 1980.

[6] D. Fensel. Triple-Space Computing: Semantic Web Services Based on Persistent Publication of Information. In *Proceedings of IFIP International Conference on Intelligence in Communication Systems*, pages 43–53, Bangkok, Thailand, November 2004.

[7] D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

[8] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler. WSMX - A Semantic Service-Oriented Architecture. In *Proceedings of IEEE International Conference on Web Services (ICWS 2005)*, pages 321–328, Orlando, FL, July 2005.

[9] R. Krummenacher, M. Hepp, A. Polleres, C. Bussler, and D. Fensel. WWW or What Is Wrong with Web Services. In *Proceedings of the 3rd European Conference on Web Services (ECOWS 2005)*, pages 235–243, Växjö, Sweden, November 2005.

[10] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara. Bringing Semantics to Web Services: The OWL-S Approach. In *Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, California, July 2004.

[11] E. Motta, J. Domingue, L. Cabral, and M. Gaspari. IRS-II: A Framework and Infrastructure for Semantic Web Services. In *Proceedings of the 2nd International Semantic Web Conference (ISWC 2003)*, pages 306–318, Sanibel Island, FL, October 2003.

[12] R. Sidhu and V. K. Prasanna. Fast Regular Expression Matching Using FPGAs. In *Proceedings of the the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2001)*, pages 227–238, Washington, DC, USA, 2001. IEEE Computer Society.

[13] W3C. Semantic Web Services Framework. Website, 2005. http://www.w3.org/Submission/SWSF.