Ontological Approach to Generating Personalized User Interfaces for Web Services

Deepali Khushraj and Ora Lassila

Nokia Research Center,
5 Wayside Road,
Burlington MA, USA

Abstract. Web services can be presented to end-users via user interfaces (UIs) that facilitate the invocation of these services. Standardized, interoperable mechanisms for describing Web service interfaces enable the generation of UIs automatically and dynamically, at least in principle; the emergence of *Semantic Web services* opens the possibility of improving the generation process. In this paper, we propose a scheme that extends the OWL-S ontology, an emerging standard for Semantic Web services, to better enable the creation of such dynamic interfaces.

Semantic Web services go beyond "classical" Web services in enabling enhanced *discovery*, *invocation* and *composition*. In our scheme, the integration of semantic descriptions of Web services with semantic models of the user's locally available data enables context-based personalization of dynamically created user interfaces, allowing us to minimize the number of necessary inputs. The need for this is compelling on mobile devices with limitations on input methods and screen size and where context data is readily available. The use of an underlying semantic model enables better accuracy than traditional form-filling techniques.

We propose an architecture for the creation and personalization of dynamic UIs from Web service descriptions. The key idea is to exploit the semantic relationships between type information of Web service input fields, and their association with information the system has about the user (such as the user's current context, PIM data, context history, usage history, corporate data etc.), in order to personalize and simplify the invocation of Web services.

1 Introduction

We observe that *Web service interfaces* [1] and *Web forms* [2, Chapter 17] bear a conceptual resemblance to one another: both specify a set of inputs and a method whose invocation yields some results. It is therefore possible to transform descriptions of Web services to form-based UIs (for invoking these services). Current formalisms for interface description, however, are not strong enough to communicate the *semantics* of services, a prerequisite for generating personalized UIs.

The *Semantic Web* is a vision of the next generation of the World Wide Web, characterized by the association of formally described semantics with content and services [3]. Work on realizing the Semantic Web is motivated by promises of greater

Y. Gil et al. (Eds.): ISWC 2005, LNCS 3729, pp. 916–927, 2005.
© Springer-Verlag Berlin Heidelberg 2005

ease – and degree – of automation, as well as improved interoperability between information systems [4]. On the Semantic Web content and services are described using representation languages such as RDF [5] and OWL [6]. Representations refer to *ontologies*, specifications of conceptualizations [7], which, in turn, enable *reasoning* via the use of logic rules.

The application of Semantic Web technologies to Web services is referred to as *Semantic Web services* [8]: Descriptions of service interfaces are associated with formal semantics, allowing *agents* to describe their functionality, discover and "understand" other agents' functionality and invoke services provided by other agents; furthermore, it is possible to combine multiple services into new services. Work on Semantic Web services is – again – driven by the possibility to automate things that formerly have required human involvement, consequently leading to improved interoperability.

OWL-S [9,10] is one of the recently emerged ontologies for semantic annotation of Web service descriptions. The OWL-S ontology is written in the ontology language OWL. Web services annotated using OWL-S can be automatically discovered, composed into new services, invoked, and their execution automatically monitored. The process model of OWL-S is used to specify how a service works by providing a semantic description of its inputs, outputs, preconditions, post conditions and process flow. The OWL-S description can be grounded to a WSDL [1] description (and possibly other standards). The grounding part of the ontology enables mapping of OWL-S inputs and outputs to the corresponding inputs and outputs in the WSDL description of the service. Hence OWL-S can be used with SOAP based Web services, which provide a WSDL description, to create Semantic Web services.

In the next section we present an approach to generating user interfaces automatically from OWL-S profiles. These interfaces can be optimized and *personalized* using semantic information about the user, collected in a *semantic cache* (as presented in section 3). Section 4 then outlines the overall architecture of our system. Finally, we present a concrete example in the form of a simple usage scenario (section 5).

2 Generating User Interfaces

Our approach is to use the OWL-S *profile* and *process model* of a service as the basic representation from which to generate a form-based UI. OWL-S provides a rich vocabulary that can be used for describing not only the (call-)interface of a service, but also other aspects that may be helpful in UI generation. There are, however, aspects of UIs that are not "derivable" from OWL-S descriptions; for this purpose, we have extended the ontology with *user interface annotations*. The extensions provide cues about:

- *display labels* used for fields,
- preferred *widget types* for implementing fields (e.g. free-text input, checkbox),
- how to render *fields with pre-determined value ranges* (e.g., a selection list) as well as the *ordering of available values* in such fields,

- *grouping* of fields and subfields, and
- how to *generate the serialized RDF data* from inputs specified by the user (the generation of serialized RDF is a requirement for invoking Semantic Web services).

(The details of the extensions are presented below; an understanding of the OWL Web ontology language as well as the OWL-S ontology is required to grasp the details.)

Every *UIModel* is associated with an OWL-S service and has associated process UIs. Multiple UIs can be attached to a single process; hence each *ProcessUI* is linked to a specific OWL-S process and a *UIFieldMap* that provides cues pertaining to the input and output fields involved in the process interaction. UI-related cues for fields are specified by creating an instance of the *UIFieldMap* class (note that a set of related fields can then be grouped together by creating an instance of the *FieldMapList*, which has individual *UIFieldMap* instances as members). Every instance of the field map can specify the following properties:

- The *parameterName* property points to the input parameter resource used in the OWL-S profile or process model. Since every input and output parameter in OWL-S has an associated parameter type, it becomes easy to identify the semantic type associated with the field using this property.
- The *parameterTypePath* property specifies a path that is used to create an OWL instance from the specified user inputs in the generated UI.
- The *fieldType* property provides cues about the type of UI widget that should be used for the given field. For example, it could specify *single select*, *multiple select*, *check box*, etc. Or, it could specify the widget type at a higher level, such as "select one" or "select many" and a widget could then be chosen at runtime based on available data about the field. Or, it could be of *FieldSet* type to specify multiple subfields. For example, a currency converter service uses inputs "price" and "currency" as input fields. Input price could further have "amount" and "currency" as subfields. The subfields are specified using the *hasSubfieldMap* property that has *FieldMapList* as range.
- The *instanceDataLocation* property along with *instanceSelectionPath* and *displayLabelPath* are used for fields that have a pre-determined value range. The *instanceDataLocation* property specifies the URL from where the value range can be found. For example, a language translator service could specify possible values for input language and output language by pointing to an ontology about languages supported by the service. Multiple locations for loading instance data can be specified using this property. The *instanceSelectionPath* property is then used to specify the path query required to select instance data from the specified data locations. Finally, the path specified by the *displayLabelPath* property is used to find the label to be used to display the instance on the UI widget.

In addition to the above properties, the UI Model fields could specify information about how conservative the UI generation scheme must be. For example, using the

UI Model, strict ordering of input fields can be imposed, or strict ordering for elements in certain selection style widgets can be imposed.

When a Semantic Web service is accessed, the associated OWL-S description along with the *UIModel* gets loaded. The rendering algorithm[1] makes use of the extended OWL-S description associated with the service to generate the UI. Once the UI is generated and inputs are received from the user, an OWL instance is created for every input parameter specified in the OWL-S description, by using the *parameterTypePath* property. The data for creating an OWL instance could be received from a single widget or from multiple widgets. The algorithm uses grouping knowledge about fields along with the *parameterTypePath* to create a single OWL instance from multiple widgets.[2] Once the OWL instances are created, the OWL-S grounding is used to invoke the service with the specified inputs. Finally, outputs of the service invocation are presented to the user. A sample *UIModel* graph associated with AltaVista's "Babel Fish" language translator Web service along with the generated UI is presented in Appendix A.

3 "Semantic Caching"

As illustrated in section 2, it is possible to generate form-based UIs from "plain" OWL-S descriptions, and potentially better ones from OWL-S profiles augmented with UI cues. Using additional information about the user – such as the current context, history of actions, etc. – allows us to further improve the generated result. The repository that stores information about the user is called the *semantic cache*. The key idea is to exploit semantic relationships between type information of Web service input (and output) fields and their association with data in the semantic cache.

The semantic cache gets data from:

- User's personal profile,
- PIM information such as address book entries, calendar entries, etc.,
- user's current context and his context history, and
- the history of inputs/outputs in recently invoked services.
- corporate data, such as company phone book, organization hierarchies etc.

The data sources for the semantic cache could go beyond the ones above. The basic requirement for any data source is that it uses a semantic model to represent data objects. In our case, we make use of semantic models that are created using the OWL Web ontology language. Making use of all the available data would significantly increase the response time for generating UIs, therefore a subset of data objects from the sources are cached.

The semantic cache stores both semantic models and data annotated with these models. A cache management algorithm constantly adds objects to and evicts them

[1] Our rendering algorithm currently generates HTML forms based on UIModels, but the renderer can easily be extended to create either XHTML or XForms based UIs.

[2] A detailed discussion of this is beyond the scope of this paper.

from the cache based on the usage patterns of the data objects in the cache. A caching scheme that implements standard caching algorithms such as MRU or LRU cannot be used directly. The scheme should also take into account the semantic relationships between objects in the cache, since the addition or deletion of a set of semantically similar objects could be done together. Additionally, the scheme should take into account the nature of data sources involved. For example, the user's current context is transient, whereas his profile information mostly remains static.

While rendering a Semantic Web service, the *type* information associated with the involved input fields is used to retrieve objects from the cache (in our case, the *parameterType* property of OWL-S provides this information). The retrieved objects are essentially instances of the class specified as the *type* of the field. By making use of a reasoner, both explicit and implicit objects of a given *type* can be queried for. The retrieved objects are given weights based on the nature of the data sources involved and their frequency of occurrence. The semantic distance between prespecified objects used by fields of the service and data in the cache can also be used to determine the relevance of a given instance. In the case of composite Web services, the relevance of a semantic instance in the cache, can further be inferred based on the atomic services that constitute the service and based on the control constructs (such as Sequence, Split etc.) specified in the service's process model. Weights are additionally adjusted based on the current context. The cumulative weight of a given object helps in determining the relevance of its use. Finally, *customizations* are made using the objects retrieved:

- *Eliminating user input widgets* for fields where the answer is already known with sufficient certainty,
- *changing UI widgets* where the input values can be predetermined (e.g. change a free-text input widget to an editable selection list),
- providing *intelligent default values* for certain fields, and
- *reordering or narrowing down* element lists in widgets such as selection lists, checkboxes, etc.

Semantic Web techniques (ontologies, reasoning, rich semantic models, etc.) can also be used in determining the user's current *usage context* and managing definitions of contexts [11]. Making the context definitions derived via semantic reasoning available to the semantic cache, and consequently to the UI generation process, can improve the system's ability to discover implicit relationships between objects in the cache. The user's context can further be applied to limit the amount of data to be considered when rendering a UI. For example, in case of *location-based services*, only data relevant to the user's current location may be considered (or at least given priority). Additionally, considering context in UI generation will improve the user's perception that the system is behaving in a *context-aware* manner.

4 Component Architecture

Figure 1 shows the component architecture of our prototype implementation. The Semantic Web services expose their service description using OWL-S and the ex-

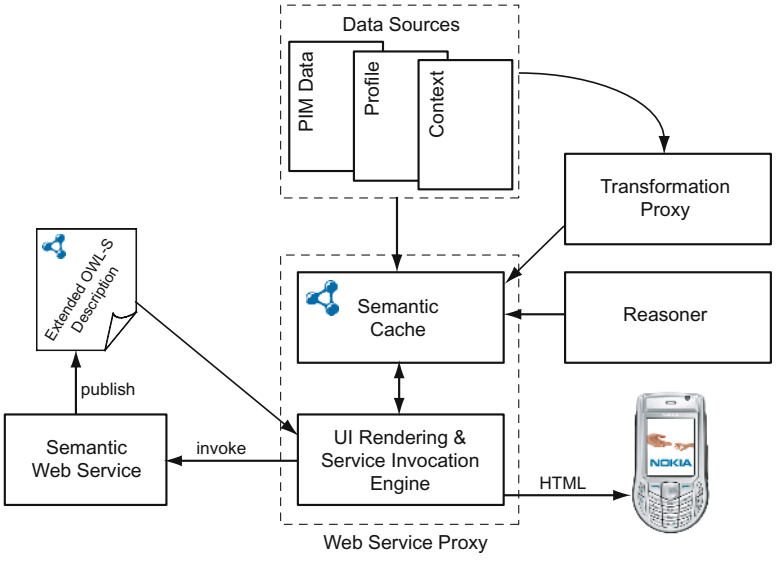

Fig. 1. System Components

tended UIModel ontology (as presented in section 2). The UIModel for a Web service could be provided by the Web service provider or by an intermediary such as an enterprise that is making services available to employees within an organization. The enterprise could thus make decisions about the allowed level of personalization, by appropriately configuring the UIModel.

When a service is to be invoked, the UI rendering engine uses the extended OWL-S description to render a dynamic web-based UI (possibly using HTML, XHTML or XForms). Additionally, it uses both explicit and implicit relationships in the semantic cache to render a personalized UI appropriate to the current context. Implicit relationships in the semantic cache are inferred using a reasoner. All data sources that feed data into the semantic cache have type information associated with them. Commonly occurring types in the semantic cache include: profile information, context history, PIM data, common sense information etc. The type information is used by the rendering algorithm to determine how relevant data from a given source is. The system can also support data originating from legacy applications using a transformational framework. In our system, the semantic cache along with the rendering engine are implemented as part of a Web service proxy. After the UI is rendered, user inputs are received to invoke the service. These inputs are used to further change the contents of the semantic cache. Finally the service is invoked and the outputs are presented as a UI.

5 Simple Usage Scenario

Our implementation was tested on a Nokia Series 60 phone with several atomic and composite real-world Web services. Example test services include, AltaVista's

"Babel Fish" language translator service, Barnes and Noble's book price finder service, zip code finder service etc. In order to test our system, the OWL-S descriptions from the *Mindswap* Web site[3] were adapted to have *UI Model* extensions. In this section, we present an example usage scenario based on some of our test services.

A mobile user visiting India is shopping for a souvenir to take back home. He makes use of the currency converter service on the phone to determine the price of the souvenir in a familiar currency. The currency converter takes three inputs: Input Price, Input Currency and Output Currency. The corresponding semantic types in the OWL-S description for each of these fields are: *XML Schema integer*, *Currency type* (represented as an OWL Class URI) and *Currency type*, respectively. The corresponding widget types in the *UI Model* are: free-text input, single select drop-down list and multiple select drop-down list, respectively (see Figure 2).

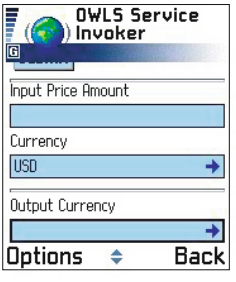

Fig. 2. Currency Converter

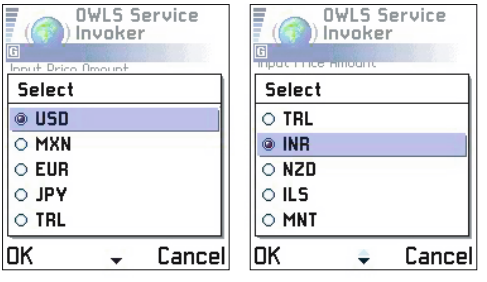

Fig. 3. Pages 1 & 2 of the currency list

Since the *Input Price* field uses a free-text input widget and has type XML Schema, only values entered by the user to invoke the same service in the recent past are used from the semantic cache. If the service was accessed recently, then the field is displayed as an editable select list, with cached values, else a free-text input widget is presented.

Since the *Input Currency* and *Output Currency* fields have the type Currency and use drop-down list widgets (with pre-determined value ranges), the currencies are ordered in the list so that relevant currencies appear on the top of the list. If any instances of Currency are found in the cache then they are likely to occur higher up on the list.

Additionally, the ordering of currencies is determined by using the semantic relationship of the Currency type class with other classes in its ontology. From the currency ontology, the rendering engine determines that every Currency object is associated with one or more countries. Hence it determines all relevant countries in the semantic cache to ascertain the ordering of currencies. In the rendered UI, USD appears high on the list because the user's profile indicates that he has an US residential address. The user's calendar information shows that he recently attended a meeting in Helsinki, his context history indicates that he recently traveled via

[3] www.mindswap.org

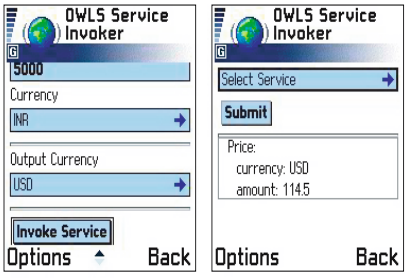

Fig. 4. Selected inputs & invocation results

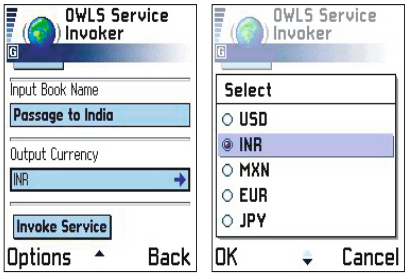

Fig. 5. Book Price Finder

Tokyo and the use of GPS coordinates suggest that the user's current location is Bombay. By using simple geo-spatial reasoning, the cache determines that the user recently attended a meeting in Finland, that he recently traveled via Japan and that he is currently located in India; hence the currencies used in these countries (i.e. EUR, JPY and INR) appear high on the list. Similarly several other countries appear high on the list based on data in the semantic cache (See Figure 3). In the current context, the relevant inputs are INR as input currency and USD as output currency (See Figure 4). Due to the reordering of currencies in the drop-down list, based on data in the semantic cache, the hassle of browsing through a long list (of 98 currencies) is avoided in this case. Once the service is invoked, the results are displayed to the user, as shown in Figure 5.

Now let us assume that the user wants to buy a book, from a local store, and check its price in local currency before he gets to the store to pick it up. The user makes use of the store's book price finder service, which takes book name and output currency as inputs. The corresponding types for these fields are: *XML Schema string* and *Currency*. Since the INR object has semantic type Currency and was recently used as an input for service invocation, it appears higher on the list, making it easier to select (see Figure 5). Note that this service was never invoked in the past, yet personalization is done based on the semantic types of fields.

Personalization of the rendered UI can further be done based on knowledge about the atomic services involved. For example, the book price finder service, presented earlier, is a composite service based on three atomic services. It first makes use of a book details grabber service, which takes a book name as input and provides the ISBN number along with other details as output. It then makes use of a book price finder service, which takes the ISBN number and provides the price in USD as output. And finally, it makes use of a currency converter service to translate the price from USD to the desired currency. In the current scenario, knowledge about the currency converter atomic service helps in further deciding the weights given to individual currencies in the drop-down list.

We observe that the use of a semantic cache for rendering personalized UIs makes form-filling easier on devices with limited text-input methods, specially for the reduction in number of keystrokes used. Furthermore, it makes it easier to personalize the UI of services that were never invoked before or that are composed of atomic services that were invoked in the past.

6 Related Work

Definition and generation of UIs for Web services has been addressed in many ways. Some of the more notable approaches include Apple's *Sherlock* application framework[4] which allows easy definition of Web service UIs using either JavaScript or XQuery, as well as various industry specifications [12,13,14]. None of these fully automates UI generation, but they are all attempts to provide a simple means of specifying UIs for pre-existing Web service interfaces. The work by Kassoff et al [15] introduces a system for *near-automatic* generation of user interfaces from WSDL profiles; in addition to a WSDL document, this system requires some additional information to generate the UI. Furthermore, the approach to providing default values requires authoring new "virtual" WSDL profiles which specify these values explicitly.

Automated form-filling techniques are often used in the context of Web-based forms. Published work in this area often addresses issues of building automated Web robots (or "bots") that need to access pages that are only reachable via various (form-based) query interfaces [16,17, for example]. Furthermore, most modern Web browsers offer some means of automatic filling or "completing" of Web forms. Although these techniques make use of personal profile information and usage history, they cannot be used for rendering personalized Semantic Web service interfaces because they will not be able to exploit the associated semantic model. Due to the lack of semantic processing capabilities, personalization based on the current context, PIM information etc. will be limited.

In database research, semantic caching techniques are frequently used to cache database queries and associated results [18, for example]. Subsequent queries are then answered by determining their semantic locality with cached queries to improve response time. As described earlier, the term "semantic caching" is used in an entirely different manner in this paper; hence semantic caching techniques for databases cannot be applied to render personalized UIs for Web services.

7 Conclusions

There is a striking resemblance between Web service descriptions and Web forms. This strongly motivates the generation of dynamic UIs for Web service access. Our work clearly establishes the need for a UI layer extension to Semantic Web service descriptions and demonstrates the benefits of semantic caching techniques for personalization of Web service UIs. The fundamental idea is to enable personalization by exploiting the relationship of semantic objects in the user's cache with type information associated with Web service inputs. Additionally, the process model associated with composite services can also be used. The use of caching is emphasized because all data about the user cannot be used while rendering personalized UIs as this would considerably increase the time required for UI generation. Semantic caching based personalization enables automatic form-filling and other customizations to UIs for services that have never been accessed before. The proposed

[4] http://developer.apple.com/macosx/sherlock/

approach has great potential for access to services from mobile devices that have limited text-input capabilities but have context information, such as current location, social context etc., readily available. Our prototype implementation uses a Web service proxy based architecture, which enables semantic processing to occur either remotely or locally on the user's mobile device. The prototype implementation was tested using several real-world Web services and an evidence to the practical benefits of the proposed approach was established. Several optimizations can be performed on the algorithm used to query for semantic objects and manage semantic objects in the cache; we would like to address this in the future.

References

1. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1. W3C Note, World Wide Web Consortium (2001)
2. Raggett, D., Hors, A.L., Jacobs, I.: HTML 4.01 Specification. W3C Recommendation, World Wide Web Consortium (1999)
3. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American **284** (2001) 34–43
4. Lassila, O.: Serendipitous Interoperability. In Eero Hyvönen, ed.: The Semantic Web Kick-off in Finland – Vision, Technologies, Research, and Applications. HIIT Publications 2002-001. University of Helsinki (2002)
5. Lassila, O., Swick, R.R.: Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, World Wide Web Consortium (1999)
6. McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language Overview. W3C Recommendation, World Wide Web Consortium (February 2004)
7. Gruber, T.R.: A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition **5** (1993) 199–220
8. Payne, T., Lassila, O.: Semantic Web Services (guest editors' introduction). IEEE Intelligent Systems **19** (2004) 14–15
9. Ankolekar, A., Burstein, M., Hobbs, J.R., Lassila, O., McDermott, D., Martin, D., McIllraith, S.A., Narayanan, S., Paolucci, M., Payne, T., Sycara, K.: DAML-S: Web Service Description for the Semantic Web. In Horrocks, I., Hendler, J., eds.: The Semantic Web - ISWC 2002. Volume 2342 of Lecture Notes in Computer Science., Springer Verlag (2002) 348–363
10. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S: Semantic Markup for Web Services. W3C Member Submission, World Wide Web Consortium (2004)
11. Khushraj, D., Lassila, O.: CALI: Context-Awareness via Logical Inference. In: ISWC 2004 workshop on Semantic Web Technology for Mobile and Ubiquitous Applications. (2004)
12. Anuff, E., Chaston, M., Moses, D., Kropp, A.: Web Service User Interface (WSUI) 1.0. Working Draft, Epicentric, Inc. (2001)
13. Kropp, A., Leue, C., Thompson, R.: Web Services for Remote Portlets Specification. OASIS Standard, Organization for the Advancement of Structured Information Standards (OASIS) (2003)

14. Arsanjani, A., Chamberlain, D., Gisolfi, D., Konuru, R., Macnaught, J., Maes, S., Merrick, R., Mundel, D., Raman, T., Ramaswamy, S., Schaeck, T., Thompson, R., Diaz, A., Lucassen, J., Wiecha, C.F.: (WSXL) Web Service Experience Language Version 2. IBM Note, IBM Corporation (2002)
15. Kassoff, M., Kato, D., Mohsin, W.: Creating GUIs for Web Services. IEEE Internet Computing **7** (2003) 66–73
16. Doorenbos, R.B., Etzioni, O., Weld, D.S.: A Scalable Comparison-Shopping Agent for the World-Wide Web. In Johnson, W.L., Hayes-Roth, B., eds.: Proceedings of the First International Conference on Autonomous Agents (Agents'97), Marina del Rey, CA, USA, ACM Press (1997) 39–48
17. Liddle, S.W., Yau, S.H., Embley, D.W.: On the Automatic Extraction of Data from the Hidden Web. In: Proceedings of the International Workshop on Data Semantics in Web Information Systems (DASWIS-2001). (2001)
18. Dar, S., Franklin, M.J., Jónsson, B.T., Srivastava, D., Tan, M.: Semantic Data Caching and Replacement. In Vijayaraman, T.M., Buchmann, A.P., Mohan, C., Sarda, N.L., eds.: VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India, Morgan Kaufmann (1996) 330–341

A Example UI Model

AltaVista's Babel Fish translator Web service is used to translate text between a variety of languages. Figure 6 represents the RDF graph for the UI model of the

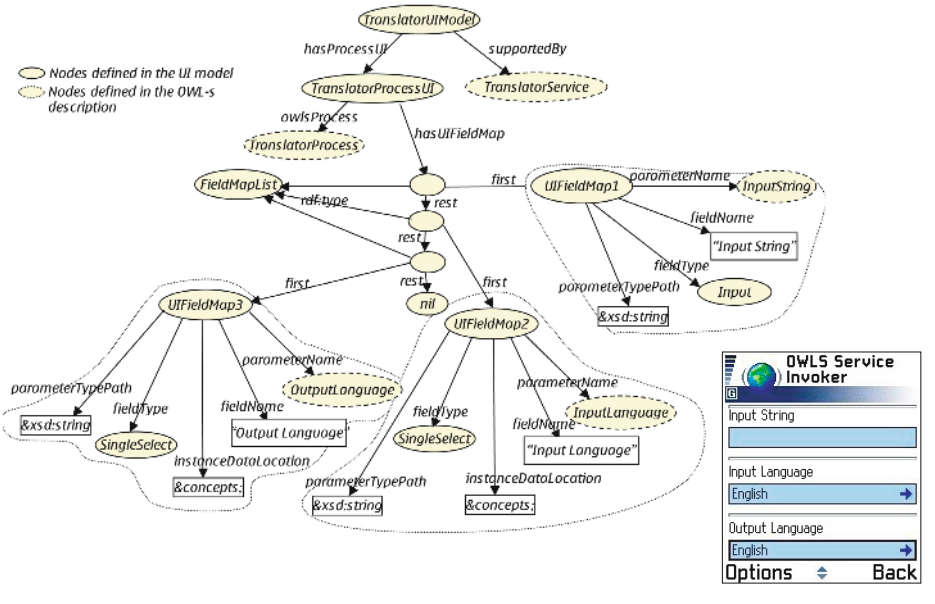

Fig. 6. Babel Fish Service & User Interface Model

service and the corresponding dynamic user interface generated by the rendering engine. The model extends from concepts defined in the OWL-S description of the service[5]. The UI model is supported by the *TranslatorService* and has an associated process UI. The specified process UI is created for the *TranslatorProcess* and has several UI fields. Each UI field is specified as a mapping between the associated parameter in the OWL-S description and other properties for rendering (and invoking) the UI. In the figure, we see three UI field map nodes, one for each of the OWL-S input parameters, namely Input String, Input Language and Output Language. A detailed explanation of all the properties emerging out of these nodes is provided in section 2.

[5] http://www.mindswap.org/2004/owl-s/1.0/BabelFishTranslator.owl