# Integrating web services into data intensive web sites

## Conference or Workshop Item

# OntoWeaver-S: Integrating Web Services into Data-Intensive Web Sites

Yuangui Lei, Enrico Motta, John Domingue
*Knowledge Media Institute*
*the Open University*
*Milton Keynes, MK7 6AA*
*{y.lei, e.motta, j.b.domingue}@open.ac.uk*

**ABSTRACT**

*Designing web sites is a complex task. Ad-hoc rapid prototyping easily leads to unsatisfactory results, e.g. poor maintainability and extensibility. However, existing web design frameworks focus exclusively on data presentation: the development of specific functionalities is still achieved through low-level programming. In this paper we address this issue by describing our work on the integration of (semantic) web services into a web design framework, OntoWeaver. The resulting architecture, OntoWeaver-S, supports rapid prototyping of service-centred data-intensive web sites, which allow access to remote web services. In particular, OntoWeaver-S is integrated with a comprehensive web service platform, IRS-II, for the specification, discovery, and execution of web services. Moreover, it employs a set of comprehensive site ontologies to model and represent all aspects of service-centred data-intensive web sites, and thus is able to offer high level support for the design and development process.*

## 1. Introduction

Designing web sites is a complex task. Ad-hoc rapid prototyping approaches easily lead to unsatisfactory results, e.g. poor maintainability and extensibility. As a consequence, a number of structured methodologies have been proposed to facilitate and guide the design and development processes [20, 8, 4, 9]. In particular, a number of methodologies and tools have been developed to support the design of data-driven web sites at a high level of abstraction. Examples include RMM [11], OOHDM [21], ARANEUS [2], WebML [5], OntoWebber [13], and HERA [7]. The key feature of these approaches is that they provide high level support for web site design, from conceptualisation and specification down to maintenance, by properly distinguishing between the different dimensions of web design and by organizing the development activities into well-structured processes. However, these approaches focus exclusively on data presentation: the development of specific functionalities is still achieved through low-level programming.

Web service technology [24, 23, 22, 10, 12] enables access to remote content and application functionalities, independently of specific implementations, or data formats. Standardised registry mechanisms (e.g., in UDDI [23]) specify the location of a service. WSDL [24] descriptions specify how to invoke a service and SOAP/XML [22] provides a uniform and standardised communication mechanisms, thus enabling easy interoperability. However, if a developer wants to bring web services into web sites, essentially he or she needs to do it entirely through low-level programming. With the partial exception of WebML [3], none of the design frameworks mentioned above provides hooks to augment web sites with functionalities, based on web service technology. Indeed, while WebML provides mechanisms to communicate with web services, the overall data-driven design framework has not been modified. For example, the user interface constructs have not been extended for enabling the access of remote web services. In other words, as far as WebML is concerned, web services are not part of the design framework; they are simply functionalities, which can be invoked, much like any other web application.

A number of web service platforms have been developed to support the design, the publication, and the invocation of web services. Examples include the Java Web Services Developer Pack (JWSDP) [12] and the IBM Web Services Toolkit (WSTK) [10]. However these platforms are very different from the web site design frameworks mentioned above. Their role is to provide powerful programming environment to build applications based on web services, rather than providing high level web site design support. Thus, we believe there is a need of integrating web service technology into high level web site design frameworks to facilitate the specification of data-intensive web sites, which allow the access to remote web services and the presentation of the results of web services. So, our goal is to define mechanisms, which allow easy integration of web services into a structured, high level support for web site design. In order to achieve this goal we make use of *semantic web services* [1, 6, 17], which rely on semantic descriptions describing the functionalities of web services in a much more powerful way than what available in WSDL/UDDI. Hence, by making use of semantic web services we can then provide high level support for bringing functionalities into a web site, by focusing the interaction with the user (i.e., the web site developer) on the semantic aspects, rather than on the technical and implementation details. In particular, we will take *OntoWeaver* [14, 15, 16], a sophisticated data-driven web design framework, as our starting point, and we will augment it to support the high level

integration of semantic web services into web sites. We will call the resulting, extended architecture, *OntoWeaver-S*.

The paper is organized as follows: section 2 describes the criteria we used to progress from OntoWeaver to OntoWeaver-S; section 3 presents the OntoWeaver-S approach to modelling the typical user interfaces of service-centred data-intensive web sites; section 4 discusses the implementation of the OntoWeaver-S mechanisms, which support the integration of web services into data-intensive web sites; and finally section 5 and section 6 describe related work and reiterate the main conclusions from this work.

## 2. From OntoWeaver to OntoWeaver-S

### 2.1. OntoWeaver

OntoWeaver approaches data-intensive web sites with special focuses on dynamic data content manipulation (i.e. data content publishing, querying, and updating), user interface composition, and customization design. In particular, it provides *a site view ontology* and *a presentation ontology* to enable the declarative representation of data-intensive web sites. Moreover, OntoWeaver proposes *a generic customization framework* for offering high level support for the specification of customization requirements and for offering comprehensive customization support for web applications at run time.

The specification of a web site in OntoWeaver consists of four perspectives:

- *The Domain Model* abstracts the back-end data sources. It contains a set of abstract concepts and relations, representing data structures of the problem domain.
- *The Site View Model* describes navigational structures for web applications and compositional user interfaces for web pages in terms of the site view ontology. In particular, the site view ontology provides a set of atomic user interface constructs as well as a set of composite user interface constructs to enable the fine-grained level support for the composition of complex user interfaces. Moreover, it provides a set of dynamic user interface to facilitate the specification of dynamic features for data-intensive web sites, including querying and manipulating the underlying domain databases.
- *The Presentation Model* specifies the visual appearance and layout for each component in the site view model in terms of the presentation ontology.
- *The Customization Model* specifies customization requirements for personalizing web pages towards individual users. It relies on the OntoWeaver customization framework, which employs a user model, a customization rule model, and a declarative site model to enable the high level support for the specification of customization requirements. OntoWeaver makes use of a customization engine to apply the specified rules to reason upon the site specifications according to the valuable user profiles (i.e. instantiations of the user models) to provide customization support.

OntoWeaver employs RDFS [19] and RDF [18] to represent ontologies and specifications. Moreover, it offers a tool infrastructure to support the entire life-cycle of a customized data-intensive web site at a high level, including modelling, design, and maintenance. More information about OntoWeaver can be found in [14, 15, 16]. Like other web modelling approaches mentioned earlier, OntoWeaver focuses on data presentation and does not offer high level support for bringing web services into web sites.

### 2.2. Evolution

The main goal of OntoWeaver-S is to provide high level support for the design of web sites that can access remote web services. Obviously, like OntoWeaver, OntoWeaver-S is also a web design framework, which provides explicit models to provide high level support for the design of web sites. Hence, the design principles of OntoWeaver-S are very much the same as OntoWeaver. For example, the modular design architecture, which distinguishes different models to approach web site design, remains the same. The presentation ontology and the customization framework do not need to be adapted either, as they abstract the common features of the presentation design and the customization design of web sites.

On the other hand, the focus of OntoWeaver-S is on the integration of web services into data-intensive web sites. Hence, in moving from OntoWeaver to OntoWeaver-S we have to introduce a number of changes:

- The site view ontology, which describes navigational structures and user interfaces, should be adapted to support the design of the site view model for service centred web sites. In particular, it should allow the high level specification of web services within the user interface elements of target web sites.
- The tool infrastructure should be adapted towards the goal of design and development of service centred web sites. In particular, the Site Designer should be modified to offer support for the design and development of this new kind of web sites. At the same time, a new run-time tool is needed to integrate web services into web sites.

As already mentioned, to enable the high level specification of web services in target web sites, OntoWeaver-S needs the support of semantic descriptions describing web services. As a result, the service layer

upon which site view models are built should be semantic. Moreover, in order to integrate web services into the target web site, OntoWeaver-S needs frameworks and tools supporting the discovery and invocation of appropriate web services by reasoning about their semantic descriptions. The IRS-II framework [17], which will be introduced in the next section, comes right in to fit the OntoWeaver-S framework.

### 2.3. Integrating Web Services into Data-Intensive Web Sites

IRS-II is an implemented infrastructure, which has been developed in our lab, the Knowledge Media Institute (http://kmi.open.ac.uk). It supports the publication, discovery, and execution of semantic web services. The following informal specification shows the task description of a semantic web service, which answers requests for flights in accordance with the given user requirements.

Task Ontology: *flight-service*
Task Name: *find-flights*
Input Roles: *from-place (type: city)*
*to-place (type: city)*
*depart-time (type: time-point)*
*arrival-time (type: time-point)*
*budget (type: amount-of-money)*
Output Roles: *flights (type: Flight)*

To invoke this semantic web service, a user simply asks for the task to be achieved in terms of the task name *find-flights* and the task ontology name *flight-service,* the IRS-II *broker* then selects an appropriate problem solving method (*PSM*) and then uses *grounding information* to locate and invoke the corresponding web service – see [17] for a detailed description of IRS-II. In particular, the input roles carry parameters for executing the corresponding web service; the output roles store the service results. Please note that IRS-II only supports one output role at the moment. The data type of the output role can be primitive e.g. String, Integer, or non-primitive, i.e. being domain classes. When the data type of the output role is not primitive, IRS-II uses XML to represent service results. For example, IRS-II uses XML to represent the results of the web service find-flights, which are instances of the class *Flight*. This class has been defined in the domain ontology of the semantic web service.

OntoWeaver-S employs IRS-II as a platform to integrate web services into data-intensive web sites. On the one hand, OntoWeaver-S relies on IRS-II to enable the access of remote web services, as IRS-II is able to locate and invoke remote web services and pass results back. On the other hand, OntoWeaver-S uses IRS-II as a platform to allow the provision of web services for data-intensive web sites.

Figure 1 shows the process of accessing remote web services in an OntoWeaver-S generated data-intensive web site. In particular, OntoWeaver-S provides a run-time tool, called *Service Integrator*, to integrate IRS-II with data-intensive web sites. Specifically, the Service Integrator collects information from a web site, then calls the IRS-II server (by means of IRS-II APIs) to invoke the specified web service and gets results from IRS-II, and finally it passes the service results back to the web site.

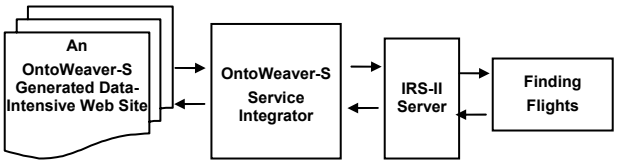


Figure 1 The process of accessing web services in OntoWeaver-S generated data-intensive web sites

OntoWeaver-S relies on a set of constructs, such as *DataComponent* and *KAComponent*, to describe the user interfaces for accessing web services and publishing results. These user interface constructs will be described in the next section.

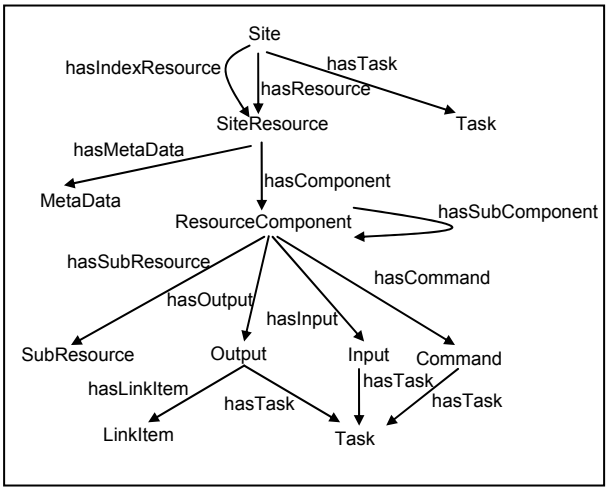


Figure 2 An overview of the OntoWeaver-S site view ontology

## 3. Modelling Typical User Interfaces for Accessing Web Services

Like OntoWeaver, OntoWeaver-S offers a site view ontology and a presentation ontology to allow the declarative representation of all aspects of service-centred data-intensive web sites. As shown in figure 2, the site view ontology consists of a set of navigational constructs, which facilitate the composition of navigational structures, and a set of user interface constructs, which support the composition of user interfaces. In particular, the site view ontology has been extended to model service-centred data-intensive web sites. Specifically, a set of user interface constructs have been extended to

allow the modelling of the typical user interfaces of service-centred data-intensive web sites, including *information visualization*, which allows the visualization of dynamic data content coming from the underlying databases or remote web services, and *information provision*, which allows the provision of information for updating or querying the back-end domain data sources or for invoking remote web services.

The navigational constructs are made up of *Site*, which models a site view as a collection of site resources i.e. web pages and a collection of task instances i.e. descriptions of semantic web services, *SiteResource*, which abstracts web pages as nodes in the navigational network and as units of user interface composition, and *LinkItem* and *DynamicLinkItem,* which model the static or dynamic link relationship between web pages. In addition, the constructs *Parameter* and *ParameterClause* are proposed to allow the specification of contextual links between pages. More information about the site structure modelling can be found in [16].

The user interface constructs can be classified into *atomic constructs*, which abstract basic user interface elements that can not be further decomposed into other elements, and *composite constructs*, which model composite user interface elements. On one hand, these user interface constructs can provide a fine grained level support for user interface composition. On the other hand, these constructs support the high level specification of access to semantic web services. Specifically, within the user interface constructs, remote web services are described in terms of *tasks*, *input roles,* and *output roles*, which comply with the semantic representation approach employed in IRS-II. Please note that the concept of output role in OntoWeaver-S is slightly different from IRS-II, when the data type of the output role of a web service is not a primitive data type but a class entity, which has a number of slots. In this case, the output roles in OntoWeaver-S refer to the slots of the result instances of the corresponding web service.

### 3.1. Information Visualization

Information visualization in service-centred data-intensive web sites presents dynamic information, which comes from the underlying databases, web resources or remote web services. OntoWeaver-S relies on the following constructs to enable the composition of the user interfaces for information visualization:

- *DynamicOutput* models the basic user interface elements that present the dynamic value of the specified slot of a given class entity or the dynamic value of the specified output role of a given web service. It has a number of attributes: the attributes *hasTask* and *hasOutputRole* are used in the case of publishing dynamic values of web services; the attributes *hasClassEntity* and *hasSlotEntity* are used to specify values from the specified slot of a class entity. The following RDF code defines one dynamic output element for publishing dynamic content of *airline*, which is one result field of the task *find-flights* (the namespace prefix '*svo*' in this paper refers to the namespace of the OntoWeaver site view ontology: xmlns:svo=”http://kmi.open.ac.uk/people/yuangui/siteviewontology#”).

```
<rdf:Description about=”datacomponet/dynamicoutput/airline” >
   <rdf:type rdf:resource=”&svo;DynamicOutput” />
        <svo:outputType>text</so:outputType>
        <svo:task rdf:resource=“find-flights” />
        <svo:outputRole rdf:resource=”find-flights/airline” />
</rdf:Descripltion>
```

- *Output* describes the basic user interface elements that present static information. Output elements are used to present explanations for dynamic values in data components.
- *OutputComponent* describes the composite user interface element for publishing the value of the specified slot of the given class entity or the value of the specified output role of the given web service. An output component typically comprises an output element, which presents explanations about the dynamic content, and a dynamic element, which displays the dynamic content.
- *DataComponent* abstracts the composite user interface elements that visualize instances of a specified class entity or results of a specified web service.

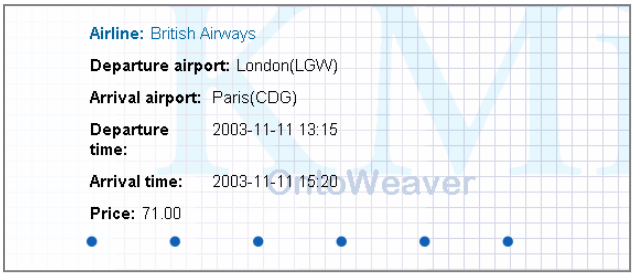


Figure 3 An user interface example for visualizing the results of the web service find-flights

Figure 3 shows an example user interface for visualizing the results of the web service find-flights. This user interface contains a number of dynamic output elements for visualizing the values of the web service find-flights. The following code illustrates the composition of this data component. Please note that at this stage, the layout of user interface elements is not considered.

```
 <!--  the composition of the entire data component -->
<rdf:Description about=”flights-result-page/datacomponent” >
  <rdf:type rdf:resource="&svo;DataComponent”/>
  <svo:task rdf:resource=“find-flights”/>
  <svo:outputComponent>
    <rdf:Bag>
      <rdf:li resource=”flights-result-page/datacomponent/airline”/>
      <rdf:li resource=”flights-result-page/datacomponent/fromairport”/>
      <rdf:li resource=”flights-result-page/datacomponent/departuretime”/>
```

```
      ……
    </rdf:Bag>
  </svo:outputComponent>
</rdf:Description>

 <!-- the composition of an output component  -->
<rdf:Description about=" flights-result-page/datacomponent/airline" >
  <rdf:type rdf:resource="&svo;OutputComponent"/>
    <!--  the output part displays explanations about the dynamic part  -->
  <svo:output>
  …
 </svo:output>
 <svo:dynamicOutput
       rdf:resource="datacomponet/dynamicoutput/airline" />
</rdf:Description>
…
```

### 3.2. Information Provision

The information provision is realized through knowledge acquisition forms, which allow users to submit information to web sites. The submitted information can be records of the underling databases or information for invoking the specified service. Please note that in this context, the service can be a built-in service provided by OntoWeaver-S or a remote web service, which has been made available through IRS-II. OntoWeaver-S provides a set of built-in services for inserting data into the underlying databases and making queries over the underlying databases. Hence, a knowledge acquisition component can be used for information provision, information query and web services access. OntoWeaver-S relies on a set of constructs to model the composition of knowledge acquisition forms:

- *Input*, which abstracts the actual input fields for allowing end users specifying information for particular slots of the specified domain class entity or for particular input roles of the specified web service. The following example defines an input element, which allows end users to enter information for the input role of *fromplace* for the web service find-flights.

```
<rdf:Description about="kacomponent/from-place/input" >
    <rdf:type rdf:resource="&svo;Input" />
    <svo:task rdf:resource="find-flights"/>
    <svo:inputRole rdf:resource="find-flights/param/from-place"/>
 </rdf:Description>
```

- *Command,* which describes the interface elements for submitting information. In particular, the definition of a command element indicates the associated service and the result page node, which intends to publish results of the associated service. As mentioned earlier, the associated service can be a built-in service for retrieving data content from the underlying databases or inserting data into the databases, or a remote web service, which has been made available through IRS-II in terms of the semantic descriptions. The following code defines a command element example for accessing the web service find-flights.

```
 <!--  the definition of the command element -->
<rdf:Description rdf:about="kacomponent/command">
   <rdf:type rdf:resource="&svo;Command"/>
   <svo:commandText>Submit</svo:commandText>
      <svo:task rdf:resource="find-flights"/>
      <svo:resultPage rdf:resource="flights-result-page" />
</rdf:Description>
```

- *InputComponent,* which describes the composite user interface elements for allowing the information provision for the specified slot of the given domain class entity or for the specified input role of the associated service. An input component typically contains an input element for presenting an input field and an output element for presenting an explanation about the input field.
- *KAComponent*, which models the composite user interface elements that present forms for achieving the functionality of information provision. A knowledge acquisition component is typically made up of a set of input components and a command element.

Figure 4 shows an example user interface for accessing the remote web service find-flights. The user interface is made up of a number of input fields and a command button for allow end users to invoke the web service. The following code illustrates the composition of this user interface.

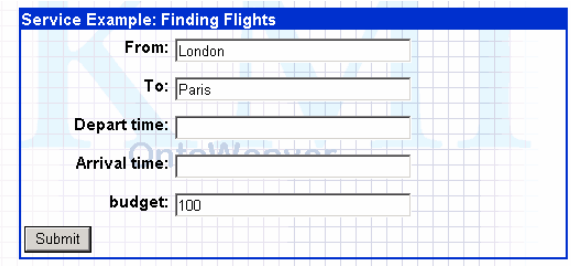


Figure 4 An user interface example for accessing the web service find-flights

```
<rdf:Description about="find-flights-page/kacomponent" >
  <rdf:type rdf:resource="&svo;KAComponent"/>
  <svo:task rdf:resource="find-flights"/>
  <svo:inputComponent>
       <rdf:Bag>
          <rdf:li resource="find-flights-page /kacomponent/from-place"/>
          <rdf:li resource=" find-flights-page /kacomponent/to-place"/>
           ……
       </rdf:Bag>
  </svo:inputComponent>
 <svo:command rdf:resource="kacomponent/command" />
</rdf:Description>

<! -- an input component example, which is composed of by a static output
element and an input element -->
<rdf:Description about="find-flights-page/kacomponent/from-place" >
  <rdf:type rdf:resource="&svo;InputComponent"/>
    <svo:output resource="kacomponet/from-place/output"/>
    <svo:input resource="kacomponet/from-place/input"/>
</rdf:Description>
…
```

## 4. Web Service Integration

In this section, we build a simple web site for accessing the example web service find-flights, which has

been discussed earlier. In particular, we create two web pages according to the following steps (please note that OntoWeaver-S offers a set of graphical tools supporting the design of web pages):

- Creating *an empty web page called find-flights-page* for holding components that allow end users to find flights according to their requirements.
- Creating *a knowledge acquisition component* in the web page find-flights-page, specifying the associated web service as the web service find-flights and choosing appropriate input roles.
- Creating *an empty web page called flights-result-page* for publishing the service result.
- Creating *a data component* in the web page flights-result-page, associating it with the web service find-flights and choosing appropriate output roles for the publication of service results.
- Specifying the web page flights-result-page as the value of the attribute *resultPage* of the command element contained in the knowledge acquisition component.

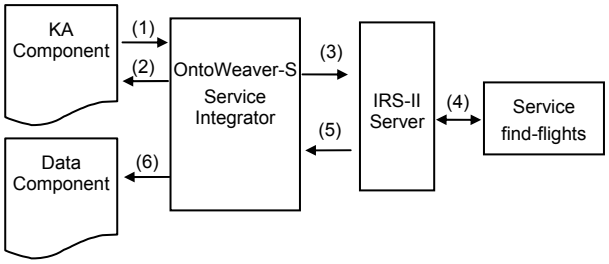


Figure 5 The process of accessing web services and publishing dynamic results coming from web services

## 4.1. Web Service Integration Process

In this section, we get a closer look at the process of accessing web services in the OntoWeaver-S generated data-intensive web sites. As illustrated in figure 5, the process of accessing web services comprises the following steps:

(1) An end user opens the web page find-flights-page, enters his or her requirements for finding flights and submits the input information to the web site.
(2) The OntoWeaver-S Service Integrator investigates the input elements, which are contained in the web page making a request for accessing the specified web service, and gathers information from the input elements for the corresponding input roles of the specified web service.
(3) The OntoWeaver-S Service Integrator calls the IRS-II Server to achieve the specified task find-flights augmented with the given constraints.
(4) The IRS-II Server invokes the corresponding web service.
(5) The IRS-II Server returns the results of the execution of web services to the OntoWeaver-S Service Integrator.
(6) The OntoWeaver-S Service Integrator passes the results back to the web page flights-result-page.

## 4.2. Implementation

In this section, we discuss the implementation issues of the integration of semantic web services into web sites. It should be noted that all code involves in this section are generated automatically by the OntoWeaver-S Site Builder during the process of compiling the declarative site specifications into web implementations. Developers neither need to cope with nor worry about the integration implementation.

During the process of compiling the OntoWeaver-S site specifications into web pages, each knowledge acquisition component is mapped to *an HTML form*. The command element contained within the component is mapped to a submit button to enable the submission of input information and the invocation of the specified semantic web service. To enable readability, the following HTML code represents a simplified JSP form, which is mapped from the knowledge acquisition component example discussed earlier. The *action attribute* of the form is mapped from the command element to indicate that the processing page for the form submission is *flights-result-page.jsp*. The *input fields* are mapped from the input elements.

```
<form action="flights-result-page.jsp" method="POST">
  <INPUT TYPE="text" name="kacomponent/from-place/input" >
  <INPUT TYPE="text" name="kacomponent/to-place/input" >
 ……
</form>
```

Each data component is mapped to an HTML table to publish dynamic content coming from the associated semantic web service. Moreover, additional server-side code is generated at the same time to enable the access of the specified web service and the publication of the service results. The following code shows the simplified JSP code generated from the definition of the data component example, which publishes results of the service find-flights. Please note that the sign of "<%" and "%" indicates that the wrapped code is server-side code, which is executed by web servers at run time.

```
 <!— Part I: instantiating a Service Integrator -- >
<jsp:useBean id="function_find_flights"
      scope="session" class="ontoweaverbean.ServiceIntegrator"/>
<jsp:setProperty  name="function_find_flights"
            property="taskName" value="find-flights" />
<jsp:setProperty name="web_site_query"
            property="taskOntologyName" value="flight-service"/>

<% //Part II: adding input roles for the specified task
function_find_flights.addInputRole("from-place",
                  request.getParameter("kacomponent/from-place/input");
function_find_flights.addInputRole("to-place",
                  request.getParameter("kacomponent/to-place/input");
……
 // Part III: achieving the specified task
function_find_flights.achieveTask(); %>
```

```
<!—part IV: presenting results of the web service  -->
<%while (function_find_flights.hasNextResultOutput())
    { %>
 <table>
      <tr><td> <%=function_find_flights.get(“airline”) %> </td></tr>
      <tr><td><%=function_find_flights.get(“fromairport”) %> </td></tr>
      ……
  </table>
 <% }%>
```

The code comprises four parts: i) the first part instantiates a service integrator using the specified task name and the task ontology name according to the specification of the associated semantic web service; ii) the second part gathers input information from the corresponding input elements and passes the information to the corresponding input roles, according to the specification of the corresponding knowledge acquisition component; iii) the third part achieves the specified task by calling the IRS-II Server; and iv) the fourth part presents the service results in a table repeatedly.

As mentioned earlier, OntoWeaver-S provides a tool called Service Integrator to integrate semantic web services into web sites. The following Java code shows how the Service Integrator achieves this goal. First, it calls the function *achieveTask()* from the IRS-II Server, augmenting the specified task name, task ontology name, and the values for the input roles. The result of this function is written in XML (this has been indicated by the semantic description of this web service). The OntoWeaver-S Service Integration then gets the result from the IRS-II Server and processes the result, and makes it ready for the presentation.

```
public void achieveTask()
{
   String serviceResult = this.irsServer.achieveTask(this.taskName,
                                        this.taskOntologyName,
                                        this.InputRoles);
   this.processingResultOutputs(serviceResult);
}

  //processing results and making it ready for publication
private void processResultOutputs(String serviceResult)
{
    ResultOutputReader reader=new ResultOutputReader(
                                            serviceResult);
   this.resultOutputList=reader.getResultOutputList();
 }

public boolean hasNextResult()
{
  boolean hasNext= this.resultOutputList.hasNext();
  if (hasNext)
      this.currentRowResult= this.resultOutputList.next();
  return hasNext;
}
  //getting the value of the specified result field of the current row
public String get(String outputName)
{
   return this.currentRowResult.get(outputName);
}
```

## 5. Related Work

Modelling approaches to web site design typically approach the design of web applications at three levels: *domain modelling*, *navigation modelling*, and *presentation modelling* [11, 2, 21, 5, 13, 7]. However, as already pointed out, these frameworks do not provide means to support access to web services. As a consequence, they limit the functionalities of the target web applications to the management of back-end data sources. WebML [5, 3] and OntoWebber [13] are the closest approaches to OntoWeaver-S.

OntoWebber defines explicit site models to abstract data-intensive web sites, and uses ontologies as the foundation for web application design. It supports the access to distributed heterogeneous data sources for the target web applications by providing wrappers and translators to process data sources in different formats. However, OntoWebber does not support the integration of (semantic) web services into web applications.

WebML relies on explicit site models to enable the design and development of data-intensive web applications. However, in comparison with OntoWeaver, the composition model, which provides means to enable the composition of user interfaces for web pages, is not expressive enough to enable the composition of complex user interface as it does not provide constructs to model atomic user interface elements. Brambillla et al. [3] have recently extended WebML by means of a set of web service hypertext primitives for communicating with web services. However, as already emphasised, they fail to integrate web services into the WebML design framework; they simply treat them as functionalities, which can be invoked, much like any other web application.

## 6. Conclusions

This paper has shown how we have augmented OntoWeaver, a *data-driven* web design framework, to produce OntoWeaver-S, a *semantic service driven* framework, which supports rapid prototyping of web service centred data-intensive web sites. OntoWeaver-S is integrated with a comprehensive platform, IRS-II for the specification, discovery, and execution of semantic web services. Moreover, OntoWeaver-S provides a set of user interface constructs to support the modelling of the typical user interfaces of service centred data-intensive web sites, including *information visualization*, which allows the visualization of dynamic data content coming from the underlying databases or remote web services, and *information provision*, which allows the provision of information for updating or querying the back-end domain data sources or for invoking remote web services.

Tools have been implemented to support service centred data-intensive web sites at design time as well as at run time. In particular, a Site Designer has been implemented to offer graphic user interfaces to allow the design of data-intensive web sites; and the Service Integrator has been prototyped to provide support for the integration of web services into data-intensive web sites at run time.

To our knowledge OntoWeaver-S is the first toolkit that attempts to integrate (semantic) web services into a high level design framework. In the future, we will focus on defining constraints validating the complex site specifications and provide tools helping developers to find and correct the specifications that are either with errors or being inconsistent in the entire site model.

## Acknowledgements

This research was partially supported by the Advanced Knowledge Technologies (AKT) project. AKT is an Interdisciplinary Research Collaboration (IRC), which is sponsored by the UK Engineering and Physical Sciences Research Council under grant number GR/N15764/01. The AKT IRC comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University.

## References

[1] A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara, DAML-S: Web Service Description for the Semantic Web, in Proceedings of the 1st International Semantic Web Conference (ISWC 2002).

[2] P. Atzeni, G. Mecca, and P. Merialdo, Design and Maintenance of Data-Intensive Web Sites, in proceeding of the 6th int. Conference on Extending Database Technology (EDBT), Valencia, Spain, March 1998.

[3] M. Brambilla, S. Ceri, S. Comai, and P. Fraternali, Model-driven Development of Web Services and Hypertext Applications, SCI2003, Orlando, Florida, July 2003.

[4] S. Ceri, P. Fraternali, and S. Paraboschi, Design Principles for Data-Intensive Web sites, SIGMOD Record, 24(1), March 1999.

[5] S. Ceri, P. Fratenali, and A. Bongio, Web Modelling Language (WebML): A Modelling Language for Designing Web Sites, WWW9 Conference, Amsterdam, May 2000.

[6] D. Fensel and C. Bussler (2002), the Web Service Modeling Framework WSMF, available online at http://informatik.uibk.ac.at/users/c70385/wese/wsmf.bis2002.pdf.

[7] F. Frasincar, G. Houben, and R. Vdovjak, Specification Framework for Engineering Adaptive Web Applications, in the Eleventh International World Wide Web Conference WWW2002 Web Engineering Track.

[8] P. Fraternali, Tools and Approaches for Developing Data-Intensive Web Applications: a survey, ACM Computing Surveys, Sept. 1999.

[9] F. Garzotto, P. Paolini, and D. Schwabe, HDM—A Model-Based Approach to Hypertext Application design, ACM Trans. Inf. Syst. 11, 1 (Jan. 1993), pp. 1 – 26.

[10] IBM Web Services Toolkit – A Showcase for Emerging Web Services Technologies, available online at http://www-3.ibm.com/software/solutions/webservices/wstk-info.html.

[11] T. Isakowitz, E.A. Stohr, and P. Balasubramaninan, RMM: A Methodology for Structured Hypermedia Design, Communications of the ACM, August 1995.

[12] Java Web Services Developer Pack, available online at http://java.sun.com/webservices/webservicespack.html.

[13] Y. Jin, S. Decker, and G. Wiederhold, OntoWebber: Model-Driven Ontology-Based Web Site Management, Semantic Web Workshop, Stanford, California, July 2001.

[14] Y. Lei, E. Motta, and J. Domingue, An Ontology-Driven Approach to Web Site Generation and Maintenance, in proceedings of 13th International Conference on Knowledge Engineering and Management, Sigüenza, Spain 1-4 October 2002, pp. 219-234.

[15] Y. Lei, E. Motta, and J. Domingue, Design of Customized Web Applications with OntoWeaver, in proceedings of the International Conference on Knowledge Capture, Florida, October, 2003.

[16] Y. Lei, E. Motta, and J. Domingue, Modelling Data-Intensive Web Sites with OntoWeaver, accepted in International Workshop on Web Information Systems Modelling (WISM 2004), Riga, Latvia, 2004.

[17] 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 2003 ( ISWC 2003), 20-23 October 2003, Sundial Resort, Sanibel Island, Florida, USA.

[18] Resource Description Framework (RDF) Model and Syntax, W3C Proposed Recommendation. http://www.w3.org/TR/PR-rdf-syntax/.

[19] Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation. http://www.w3.org/TR/rdf-schema/.

[20] W. Retschitzegger, W. Schwinger, Towards Modelling of DataWeb Applications - A Requirement's Perspective, Proc. of the Americas Conference on Information Systems (AMCIS) Long Beach California, Vol. I, August 2000.

[21] D. Schwabe and G. Rossi, The Object Oriented Hypermedia Design Model, Comm. Of the ACM, Vol.38, #8, pp 45-46, August 1995.

[22] Simple Object Access Protocol (SOAP) (2000). W3C Note 08. Available online at http://www.w3.org/TR/SOAP/.

[23] UDDI Specification, available online at http://www.uddi.org/specification.html.

[24] Web Services Description Language (WSDL) (2001), W3C Note 15, available online at http://www.w3.org/TR/wsdl.