Modelling Data-Intensive Web Sites with OntoWeaver

Yuangui Lei, Enrico Motta and John Domingue

Knowledge Media Institute
The Open University
Walton Hall, MK7 6AA
{y.lei, e.motta, j.b.domingue}@open.ac.uk

Abstract. This paper illustrates the OntoWeaver modelling approach, which relies on a set of comprehensive site ontologies to model all aspects of data-intensive web sites and thus offers high level support for the design and development of data-intensive web sites. In particular, the OntoWeaver site ontologies comprise two components: a site view ontology and a presentation ontology. The site view ontology provides meta-models to allow for the composition of sophisticated site views, which allow end users to navigate and manipulate the underlying domain databases. The presentation ontology abstracts the look and feel for site views and makes it possible for the visual appearance and layout to be specified at a high level of abstraction.

1 Introduction

Building a data-intensive web site is a complex task. An ad-hoc development methodology often results in few re-usable components, costly and time consuming development processes, and poor performance on maintenance. The situation becomes even more difficult when customization issues arise and web sites need to be dynamically adapted to different users. To address this problem, a number of model-based approaches have been proposed recently, which facilitate and guide the activities of specification and refinement of all aspects of web applications [6, 15, 10, 1, 9, 3, 11, 5]. The key feature of these approaches is that they provide high level support for web site design, from conceptualization and specification down to maintenance. However, these approaches do not provide expressive constructs, which are powerful enough to describe complex user interfaces for web sites. For example, some approaches, e.g. HDM [6] and RMM [10], only provide navigational constructs to allow the description of navigational structures; the composition of sophisticated user interfaces is beyond their considerations. Other approaches, like WebML [3], OntoWebber [11] and HERA [5], do provide user interface constructs to model the composition of user interfaces. However, their constructs are defined at a coarse-grained level. As a consequence, web developers cannot use high-level model-based approaches in the creation of the user interfaces. Moreover, as web sites are not entirely represented declaratively, the functionalities of intelligent analysis and management over web sites cannot be supported appropriately. For example, although customization has been taken into consideration in approaches like OOHDM [15], UWE [9], WebML [3] and HERA [5], none of these approaches provide extensive

customization support by reasoning upon the entire site model, including site structure personalization, user interface personalization and domain data content customization.

To address these issues, OntoWeaver is proposed to employ the notion of *ontology* [8] as the backbone to provide comprehensive support for the design and management of customizable data-intensive web sites [12, 13]. In particular, OntoWeaver relies on a set of comprehensive *site ontologies* to enable the declarative representation of all aspects of data-intensive web sites. In this way, OntoWeaver is able to offer high level support for specifying data-intensive web sites. Furthermore, the idea of working with declarative specifications of all aspects of a web site opens up a number of possibilities with respect to intelligent analysis and management. Customization and site design critiquing are such examples. Specifically, as the entire site model is available to customization, the OntoWeaver support for customization is not restricted in the way existing approaches are. Moreover, recommendations can be produced for web developers to improve their design by applying a set of critiquing rules to reason upon the entire site model.

This paper illustrates the OntoWeaver approach to modelling data-intensive web sites. It is organized as follows: section 2 presents an overview of the OntoWeaver methodology; section 3 explains the OntoWeaver approach to modelling data-intensive web sites in detail; section 4 describes the related work; and finally section 5 concludes our work and presents the future work.

2 An OntoWeaver Overview

An ontology is an explicit formal specification of the terms in the domain and relations among them [8]. It provides a common understanding of topics that can be communicated between people and application systems, and thus envisions the next generation of the Web, Semantic Web [2]. In the context of web site design, an ontology-based approach provides explicit vocabularies for specifying the target web sites in an exchangeable format and thus enables the management and maintenance to be carried out at the knowledge-level. Moreover, it results in declarative representation of target web sites, which in turn opens up a number of possibilities with respect to intelligent analysis and management of web sites.

OntoWeaver provides a set of comprehensive site ontologies to model all aspects of data-intensive web sites. Specifically, *a site view ontology* is proposed to model site structures and user interfaces; *a presentation ontology* is defined to describe the presentation styles and layouts for web pages. Moreover, *a generic customization framework* is proposed to offer high level support for customization design. With OntoWeaver, the specification of a data-intensive web site comprises the following components:

- *A domain ontology*, which abstracts the back-end data sources. It contains *concepts* and *relations* between these concepts. The domain data content is considered as instances of domain concepts. The relations between concepts can be used to facilitate the navigation design.
- *A domain knowledge base*, which describes domain data content in terms of the domain ontology.

- *Site view specifications*, which describe navigational structures for web applications and user interfaces for web pages.
- *Presentation specifications*, which express the presentation styles and layouts for components contained in the site view specifications.
- *A user ontology*, which models information about end users.
- *A user model*, which describes user profiles in terms of the user ontology.
- *Customization rules*, which define the rules for personalizing the generic view of a web application towards particular users or user environments.

OntoWeaver prescribes the semantic web standard RDF [16] and RDFS [17] to represent all aspects of web sites. Nevertheless, there are several limitations with respect to the expressiveness of the RDF Schema language. For example, it is not possible to specify cardinalities for properties. OntoWeaver relies on its tool suite to solve this problem by means of internal models. In the future we plan to use the new emerging semantic web standard OWL [18] as the underlying language to represent ontologies and specifications (i.e. populated ontologies).

2.1 Customization Support

OntoWeaver strictly separates the domain data model, the site view model and the presentation model. This architecture guarantees design time customization support. Essentially site developers can make use of this modular approach to define: (1) at the site view level, different site views over the same domain model for different user groups or different types of devices, and (2) at the presentation level, different layouts and appearances for the same site view thus giving flexibility for the requirements of different user groups. However, customization should also be supported dynamically according to the contextual information of each user individual. To this purpose, OntoWeaver proposes a customization framework, which makes use of *a user model* and *a customization rule model* to enable the high level support for extensive customization design. Moreover the customization framework employs *a customization engine* to reason upon the site specifications to provide customization support for the target web sites at run time. More information about the customization framework can be found in [12].

2.2 Site Generation

The idea of using ontologies to drive software generation has been demonstrated in the area of knowledge acquisition, where a number of meta-tools have been developed, which employ domain ontologies to drive the generation of knowledge acquisition tools [4] [7] [14]. The major advantage of this methodology is that developers, especially domain experts, can specify software tools readily, and that a pre-existing ontology can be used as the basis for the specifications.

As shown in figure 1, OntoWeaver employs the same methodology to drive the generation of data-intensive web sites. In particular, it relies on the domain ontologies and the site ontologies to declaratively represent all aspects of the target software –

data-intensive web sites, and thus enables the high level support for design and maintenance. With OntoWeaver, the web site generation process involves in the following steps:

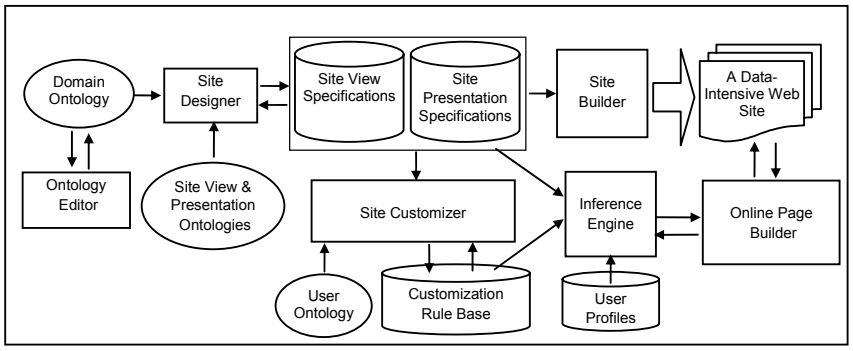

Fig.1. The OntoWeaver Framework

- Mapping the domain ontology to the site view ontology to create a site view specification, describing complex navigational structures and sophisticated user interfaces for allowing end users to navigate and manipulate the underlying dynamic data content.
- Mapping the site view ontology to the presentation ontology to create a presentation specification. This process involves in specifying presentation styles and layouts for each user interface element contained in the site view specification.
- Compiling the site view specification and the presentation specifications to generate data-intensive web site implementations. In particular, different presentation specifications can be augmented for the same site view specification to generate different site implementations. Customized web pages are generated on the fly at run time according to the result of the customization inference, which applies the specified customization rules and the user profiles to reason upon the declarative site specifications.

As shown in figure 1, OntoWeaver offers a set of tools to support the design and development of data-intensive web sites, including *an Ontology Editor,* which allows developers to edit ontologies, *a Site Designer*, which supports the design of a data-intensive web site, *a Site Builder*, which compiles site specifications into web site implementations, *a Site Customizer*, which supports the customization design, *a Customization Engine*, which performs inferences upon the site specifications, and *an Online Page Builder*, which generates customized web pages on the fly according to the result of the customization engine.

3 Modelling Data-Intensive Web Sites

In this section, we illustrate the OntoWeaver approach to modelling data-intensive web sites from the following perspectives: the support for composing sophisticated

site structures, the support for composing sophisticated user interfaces and the support for specifying complex layouts and visual appearances.

3.1 An Example

Figure 2 shows a domain ontology, which abstracts the data model for the KMi Web Portal, a data-intensive web site example, which publishes information about our lab - the Knowledge Media Institute within the Open University. There are two relationships between the domain entities: i) *the hierarchy relationship* (i.e. *is-a*), which has been shown using arrows in figure 2(a), and ii) *the has-a relationship*, which is the relationship between class entities and property entities, as shown in figure 2(b).

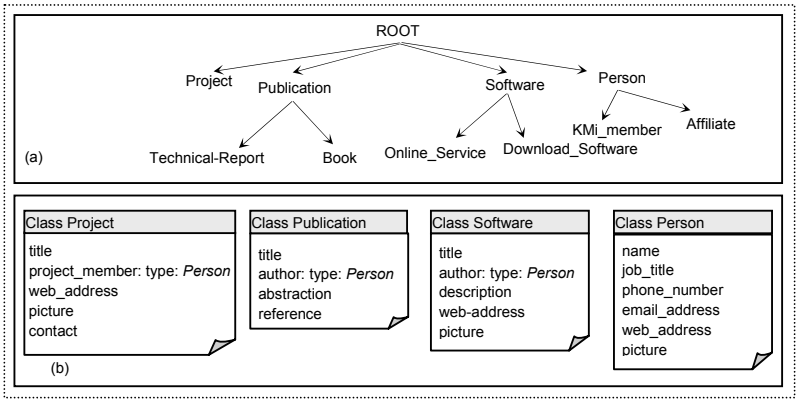

Fig.2. A domain ontology example. Part (a) shows the hierarchy structure of the domain ontology. Part (b) shows the descriptions about some of the classes.

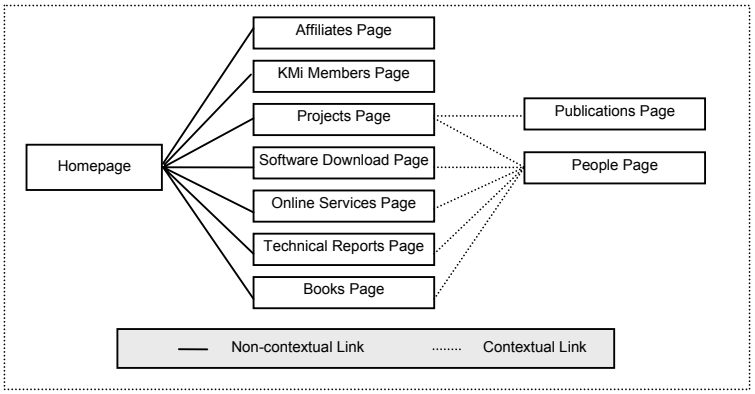

Fig.3. A site structure example of the KMi Web Portal

Figure 3 shows an example site structure for the KMi Web Portal, which comprises a number of web pages and a number of links. Each web page has its own purpose for publishing particular information; each link facilitates navigation from one web page to another. In particular, there are *contextual links* and *non-contextual links*. The contextual links carry the contextual information from the source web page to the destination web page. For example, the link between the project web page and the people web page is contextual, as it allows navigation from the brief information of project members to the detailed information of the corresponding person. The non-contextual links are on the opposite, which do not carry any information along with the navigation.

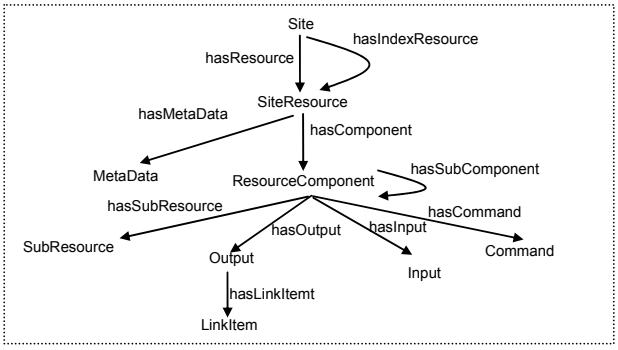

Fig.4. An overview of the OntoWeaver site view ontology

3.2 The Site View Ontology

Figure 4 shows an overview of the site view ontology: it models a web site as a collection of logical resources; the logical resources describe web pages and are abstracted as compositions of resource components; and the resource components are described as compositions of a number of user interface elements, e.g. output elements, input elements, command elements or sub resource components.

The site view ontology offers *a set of navigational constructs* to facilitate the specification of complex navigational structures and *a set of user interface constructs* to allow for the composition of sophisticated user interfaces. The user interface constructs can be further classified into *atomic user interface constructs*, which describe atomic user interface elements that can not be further decomposed into other elements, and *composite user interface constructs*, which describe composite user interface elements. Details of these constructs are shown in table 1.

Table 1. The constructs of the site view ontology

Construct	Sub Constructs	Slots	Description
Composite User Interface Constructs			
Site		• hasIndexResource • hasResources • hasDomainURI	Modelling web sites as a collection of web resources.

SiteResource		• hasComponent • hasMetaData	Modelling web pages.
ResourceComponent	• KAComponent • DataComponent • SearchComponent • OutputComponent • InputComponent	• hasSubComponent • hasOutput • hasInput • hasCommand • hasSubResource	Modelling user interface elements that compose web pages.
KAComponent		• hasClassEntity • hasInputComponent • hasCommand	Modelling components that allow users to input facts to the underlying databases.
DataComponent		• hasClassEntity • hasOutputComponent	Modelling components that publish data content coming from the underlying databases.
SearchComponent		• hasClassEntity • hasInputComponent • hasCommand	Describing components that allow users to make queries over the back-end databases.
OutputComponent		• hasOutput • hasDynamicOutput	Publishing dynamic domain content retrieved from the specified slot of the specified domain class entity.
InputComponent		• hasOutput • hasInput	Presenting interface elements to gather input from end users for a particular slot of the specified class entity.
Atomic User Interface Constructs			
Input		• hasClassEntity • hasSlotEntity	Expressing the basic interface elements that allow end users entering meaningful information to interact with web applications.
Output	• DynamicOutput	• hasOutputType • hasOutputValue • hasLinkItem	Abstracting the basic elements that present a piece of information, either being static/dynamic, text/image, plain or having links associated with it.
DynamicOutput		• hasClassEntity • hasSlotEntity	Displaying dynamic content coming from the specified slot of the specified class entity.
Command		• hasService • hasResultPage	Abstracting the basic interface elements that enable end users to invoke the specified services and bring dynamic content to users.
SubResource		• associatedResourceURI • isExtenralResource	Expressing the basic elements, which import specified web pages into web pages.
Navigational Constructs			
LinkItem	• DynamicLinkItem	• associatedResourceURI • isExternalResource • hasParameter	Modelling link relationships.
DynamicLinkItem		• hasClassEntity • hasSlotEntity	Modelling the link items that come from the underlying knowledge bases.
Parameter		• hasParamterClause	Describing contextual information which flows along with links.
ParameterClause		• hasClassEntity • hasSlotEntity • hasRelationOperator • hasValue • hasLogicalOperator	Abstracting parameter clauses, which compose parameters.
MetaData		• hasPageHeadline • hasIntroduction • hasDescription • hasAuthors	Describing meta-information for web pages.

3.3 Modelling Site Views

With OntoWeaver, modelling site views is achieved through two steps: i) defining site structures for the target web site, i.e. identifying web pages and defining their purposes and link relationships, and ii) composing detailed user interfaces for each web page. In the next sub-sections, we will explain these processes.

3.3.1 Modelling Site Structures

By the term of site structure, we describe a coarse-grained level structure of an entire web site, which comprises an index page, a number of page nodes and the URI of the underlying domain ontology. OntoWeaver relies on the construct *Site* to describe the components of web sites, the construct *SiteResource* to specify the initial purpose of each page node and the construct *LinkItem* to express link relationships between page nodes. The following RDF [16] code defines a site structure for the KMi Web Portal (The prefix '*svo*' refers to the namespace of the OntoWeaver site view ontology: xmlns:svo=”http://kmi.open.ac.uk/people/yuangui/siteviewontology#”).

```
<rdf:Description rdf:about="http://localhost:8080/kmiportal">
   <rdf:type rdf:resource=”&svo;Site” />
   <svo:indexResource rdf:resource="kmiportal/indexpage"/>
   <svo:domainOntologyURI>http://kmi.open.ac.uk/yuangui/kmiontology </svo:domainOntologyURI>
   <svo:siteResource>
      <rdf:Bag>
           <rdf:li rdf:resource="kmiportal/people/affiliate" />
            …
     </rdf:Bag>
  </svo:siteResource>
</rdf:Description>

<rdf:Description rdf:about="kmiporta/indexpage" >
  <rdf:type rdf:resource=”&svo;SiteResource” />
   …
</rdf:Description>
```

Each page node is defined as an instance of the construct SiteResource. At the coarse-grained level, the detailed content of page node is not considered. But the purpose (e.g. headlines and descriptions) of each page node is identified and specified through the construct MetaData. The link relationships between the page nodes are defined within the specification of page nodes. At the coarse-grained level, as the physical locations of links are not concerned, all links are placed in a default component, called navigation component, which can be edited at the web page composition stage. The following code specifies the initial content for the index page, which contains meta-data and a navigation component, which further consists of a number of links enabling the navigation from the index page to other web pages.

```
<rdf:Description rdf:about=”kmiportal/indexpage” >
   <rdf:type rdf:resource=”&svo;SiteResource” />
   <so:metaData rdf:resource="kmiporta/indexpage/MetaData"/>
   <so:resourceComponent rdf:resource=”kmiportal/indexpage/navigationcomponent” />
</rdf:Description>
```

Links are defined using the constructs *Output* and *LinkItem*. An Output instance is a user interface element, which presents a piece of information. The links are output elements, which are associated with link items. The following code defines a link example, which allows the navigation from the web page in question to the KMi_member page.

```
<rdf:Description rdf:about="kmiportal /indexpage/navigationcomponent/kmi_member">
   <rdf:type rdf:resource="&svo;Output"/>
   …
   <svo:linkItem>
       <rdf:Description rdf:about="kmiporta/indexpage/navigationcomponent/kmi_member/link ">
           <svo:associatedResourceURI>kmiportal/kmi_member</svo:associatedResourceURI>
           <svo:isExternalResource>false</svo:isExternalResource>
```

```
        </rdf:Description>
    </svo:linkItem>
</rdf:Description>
```

3.3.2 Composing User Interfaces for Web Pages

With OntoWeaver, web pages are composed of by a set of components, and each component is further composed of a set of atomic user interface elements and sub-components. In this way, complex user interfaces can be easily expressed. Figure 5 shows an example user interface for the index page of the KMi Web Portal. It is composed by a number of components: a navigational component presenting hyperlinks, a component displaying a headline and a component presenting an introduction. Each component is further made up of a number of sub-elements. The following code illustrates the way of composing user interface elements in OntoWeaver. Please note that the organization and look and feel of the user interface elements are not considered at the stage of user interface composition.

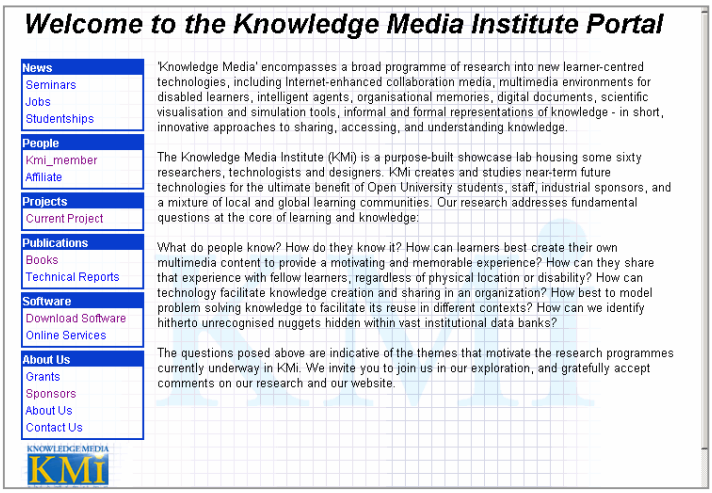

Fig.5. An example user interface for the index page of KMi Web Portal

```
<rdf:Description rdf:about=”kmiporta/indexpage” >
  <rdf:type rdf:resource="&svo;SiteResource"/>
  <svo:resourceComponent>
    <rdf:Bag>
       <rdf:li rdf:resource==”kmiporta/indexpage/navigationcomponent” />
      <rdf:li rdf:resource==”kmiporta/indexpage/headline” />
      <rdf:li rdf:resource==”kmiporta/indexpage/introduction” />
    </rdf:Bag>
  </svo:resourceComponent>
</rdf:Descrption>

<rdf:Description rdf:about=”kmiporta/indexpage/headline” >
  <rdf:type rdf:resource="&svo;ResourceComponent"/>
    <svo:output>
          <rdf:Bag>
             …
           </rdf:Bag>
```

```
    </svo:output>
</rdf:Description>
…
```

With OntoWeaver, user interface elements can be re-used, as each user interface element has a URI to identify itself and can be referenced in any composite user interface elements. Hence, OntoWeaver provides a cost-effective way for managing and maintaining the user interfaces of web pages. For example, in the KMi Web Portal, the navigation component of the index page is re-used in all other page nodes. Thus, each web page shares the same navigation pattern. Any maintenance or modification on the navigation component is only done once and takes effects in all web pages.

3.3.3 Composing Dynamic User Interfaces

The dynamic features of data-intensive web sites include i) information publication, which publishes the dynamic data content coming from the back-end knowledge base, ii) information provision, which allows end users to submit information to the domain knowledge base, and iii) information querying, which allows end users to search information. OntoWeaver provides a set of user interface constructs to support for the composition of user interfaces for realizing these dynamic features.

Information Publication. Data content publication can be specified by means of the construct *DataComponent*. The definition of a data component requires i) a class entity whose instances are to be published and ii) a list of output components, which compose a user interface for the data component publishing the values of the specified slot entities for the specified class entity. In particular, an output component presents the dynamic value of a particular slot of the specified class entity. It typically contains a static output element, which presents an explanation about the dynamic value, and a dynamic output element, which presents dynamic value retrieved from the underlying domain databases. The following code illustrates the example data component, which presents instances of the class Project and has been shown in figure 6.

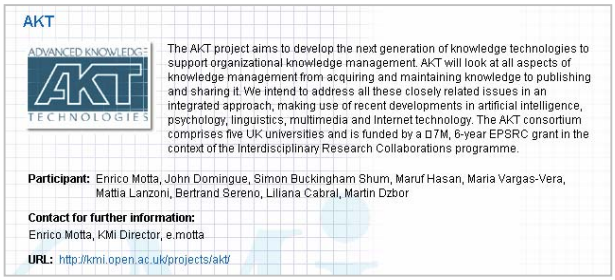

Fig.6. An example user interface for information publication

```
<rdf:Description about="kmiportal/projectpage/dataComponent" >
  <rdf:type rdf:resource="&svo;DataComponent"/>
  <svo:classEntity rdf:resource="Project" />
  <svo:outputComponent>
```

```
    <rdf:Bag>
      <rdf:li rdf:resource="kmiportal/projectpage/dataComponent/contact" />
        …
      </rdf:Bag>
  </svo:outputComponent>
</rdf:Description>

<rdf:Description about="kmiportal/projectpage/dataComponent/contact" >
  <rdf:type rdf:resource="&svo;OutputComponent" />
  <svo:output>
      … <!-- present explanation -->
  </svo:output>
  <svo:dynamiceOutput>
     <rdf:Description about="kmiportal/projectpage/dataComponent/contact/dynamicoutput">
            <svo:outputType>text</svo:outputType>
            <svo:classEntity rdf:resource="Project"/>
           <svo:slotEntity rdf:resource="contact"/>
      </rdf:Descripltion>
  </svo:dynamicOutput>
</rdf:Description>
```

Information Provision and Information Querying. The information provision and information querying are realized through forms, which allow users to submit information to data-intensive web sites. OntoWeaver provides a construct *KAComponent* to support the specification of user interfaces for updating the underlying knowledge base and a construct *SearchComponent* to enable the composition of user interfaces for querying the knowledge base. To support these user interfaces, OntoWeaver defines a set of services for enabling the manipulation of the back-end databases, including data content updating and querying.

A knowledge acquisition component, an instance of KAComponent, presents a form to allow the gathering of information from end users and the creation of instances of the specified domain class entity. The user interface of a knowledge acquisition component comprises *a list of input components*, which specifies the slots whose values are going to be used for creating new facts for the specified class entity, and *a submit command*, which allows users to submit information and invoke the corresponding knowledge acquisition operation.

A search component presents a form to allow the gathering of information from end users and making queries over the domain knowledge base. The user interface of a search component contains a class entity, which specifies the class entity that the search component works on, a list of input components, which allow end users to enter information as the foundation of carrying out queries, and a search command, which specifies the pre-defined search service and the place to publish the search results. In particular, the input component part specifies the slots that are used to compose queries. The following code illustrates the search component example, which allows end users to search information about the specified person. There is only one input component in this search component, which means that the queries of this user interface element are based on one particular slot of the class Person.

```
<rdf:Description about="kmiportal/indexpage/searchcomponent" >
  <rdf:type rdf:resource="&svo;SearchComponent"/>
  <svo:classEntity rdf:resource="Person" />
  <svo:inputComponent>
       <rdf:Bag>
           <rdf:li resource="kmiportal/indexpage/searchcomponent/personname" />
       </rdf:Bag>
```

```
  </svo:inputComponent>
 <svo:command>
     <rdf:Description rdf:about=”kmiportal/indexpage/searchcomponent/command” >
        <svo:serviceName> search_instance </svo:serviceName>
        <svo:resultPage>kmiportal/search/resultpage</svo:resultPage>
     </rdf:Description>
 </svo:command>
</rdf:Description>

<rdf:Description about="kmiportal/indexpage/searchcomponent/inputcomponent/name” >
<rdf:type rdf:resource="&svo;InputComponent"/>
  <svo:output>
      …
   </svo:output>
  <svo:input>
     <rdf:Description rdf:about=" kmiportal/indexpage/searchcomponent/inputcomponent/name/input”>
         <svo:classEntity rdf:resource="Person"/>
         <svo:slotEntity rdf:resource="person_name"/>
     </rdf:Description>
   </svo:input>
</rdf:Description>
```

3.3.4 Defining Links

In OntoWeaver, link items contribute to the user interface composition by providing a mechanism to define hyperlinks for output elements. Link items in web sites can be non-contextual and contextual. The non-contextual links have been illustrated in section 3.3.1. The contextual link requires the correct information flow between the source web page and the destination web page. It typically constraints data content of the destination web page.

OntoWeaver relies on the construct *Parameter* and the construct *ParameterClause* to enable the specification of contextual links. The definition of a parameter clause comprises five parts: *a class entity* and *a slot entity*, which value is going to be constrained in the contextual information flow, *a relation operator* e.g. “equal” and “not equal”, which defines the method to constrain data content, *a value*, which is used to constrain data content, and *a logical operator*, which indicates the relation of the corresponding parameter clause with the rest of the parameter clauses. A parameter can be made up of a set of parameter clauses by means of logic operators e.g. *AND* and *OR*.

The following code defines a contextual link example, which is associated with a dynamic output element displaying project members for the project instances. This contextual link allows the navigation from the project page to the people web page to present the detailed information about the corresponding person. As we can see from the definition, the value of the link parameter is the same as the dynamic value of the output element, as the dynamic value indicates the contextual information flowing with the link.

```
<rdf:Description about=”kmiportal/projectpage/dataComponent/members” >
     <rdf:type rdf:resource="&svo;DynamicOutput"/>
      …
       <svo:linkItem rdf:resource="kmiportal/projectpage/dataComponent/members/linkitem" />
</rdf:Descripltion>

<rdf:Description rdf:about="kmiportal/projectpage/dataComponent/members/linkitem">
      …
       <svo:parameter rdf:resource="kmiportal/projectpage/dataComponent/members/linkitem/parameter" />
</rdf:Description>
```

```
<rdf:Description rdf:about="kmiportal/projectpage/dataComponent/members/linkitem/parameter">
  <rdf:type rdf:resource="&svo;Parameter"/>
  <svo:parameterClause>
   <rdf:Description rdf:about="kmiportal/projectpage/dataComponent/members/linkitem/parameter/clause" >
       <svo:classEntityURI>Person</svo:classEntityURI>
       <svo:slotEntityURI>person_name</svo:slotEntityURI>
       <svo:relationOperator>EQUAL</svo:relationOperator>
       <svo:value>parent.outputvalue</svo:value>
   </rdf:Description>
  </svo:parameterClause>
</rdf:Description>
```

3.4 Modelling Visual Appearances and Layouts

The visual appearance and layout issues have been ignored in most web modelling approaches [6, 10, 1, 5, 3]. As a consequence, web developers still need to do a lot of low-level work to implement visual appearance and layout for web sites. To address this problem, OntoWeaver proposes a presentation ontology, which relies on a set of *template constructs* to abstract visual appearances of user interface elements, a set of *layout constructs* to model the organization features of user interface elements, a construct called *Presentation* to attach templates to user interface elements and a construct called *SitePresentation* to group the specifications of presentation styles and layouts together as a complete presentation model.

OntoWeaver employs a template-based approach to specifying visual appearances for user interface elements. This approach enables the re-use of the visual appearance specification, and thus helps developers to specify consistent visual appearances for web pages.

3.4.1 Specifying Layouts

OntoWeaver defines two layout constructs to model typical layout for user interface elements: i) *TextLayout* models layout of atomic user interface elements in terms of *alignment*, which describes the alignment of a user interface element within a component, and ii) *ComponentLayout* abstracts the organization features of composite interface elements. In particular, a component layout organizes the sub-elements of its corresponding component into five sub areas, which are *top*, *left*, *middle*, *right* and *bottom*. Each area can display a number of user interface elements in a specified layout direction, i.e. horizontal direction or vertical direction. The layout definition of each area is abstracted by means of the construct *ComponentAreaLayout,* which relies on *a list of siteEntityURIs* to indicate the user interface elements presented in the corresponding area, a property called *layoutDirection* to describe the direction of presenting the specified user interface elements, and a property called *areaSize* to specify the size of the sub area.

Figure 6 has shown a layout example for publishing instances of the class Project: i) the *title* component is placed in the top area; ii) the *picture* component is placed in the left area; iii) the *description* component is placed in the middle area; and iv) all other components are put in the bottom area and arranged together vertically. In addition, the area size can be adjusted for each area. Moreover, this specification only defines the organization of the components contained in the data component; the

further layout within the sub components can be defined in the same way, and thus allows the specification of sophisticated layouts for user interface components. However, because of the limited space we only show a simplified specification of this layout example (The prefix '*spo*' refers to the namespace of the OntoWeaver site presentation ontology: xmlns:spo=”http://kmi.open.ac.uk/people/yuangui/sitepresentationontology#”).

```
<rdf:Description rdf:about="kmiportalprojectpage/dataComponent/layout" >
    <rdf:type rdf:resource=”&spo;ComponentLayout" />
    <spo:siteEntityURI>kmiportal/projectpage/dataComponent</spo:siteEntityURI>
    …
   <spo:areaLayout rdf:about=“kmiportal/projectpage/dataComponent/arealayout/bottomarea" />
</rdf:Description>
  <!—the specification for the bottom area -->
 <rdf:Description rdf:about= “kmiportal/ projectpage/dataComponent/arealayout/bottomarea" >
      <rdf:type rdf:resource=”&spo;CompoentAreaLayout” />
      <spo:areaType>Bottom Area</spo:areaType>
      <spo:layoutDirection>Vertical</spo:layoutDirection>
      <spo:siteEntityURI>
        <rdf:Bag>
            <rdf:li>kmiportal/projectpage/dataComponent/outputcomponent/members</rdf:li>
            <rdf:li>kmiportal/projectpage/dataComponent/outputcomponent/contact</rdf:li>
            …
        </rdf:Bag>
    </spo:siteEntityURI>
</rdf:Description>
```

4 Related Work

Recently, a number of tools and approaches have been developed to address the design and development of data-intensive web sites. Examples include RMM [10], OOHDM [15], ARANEUS [1], WebML [3], OntoWebber [11] and HERA [5]. These approaches have addressed the ability to model the underlying domain data structures and handle the dynamic data content. This is typically achieved by allowing the abstraction of domain data structures and by allowing the specification of user interfaces for accessing the underlying domain data. In particular, like OntoWeaver, domain data content in WebML [3] is not only readable, but also manageable, as WebML has defined a set of operation constructs, which support the management of the underlying domain data.

Regarding the support for modelling sophisticated site views, the approaches mentioned above only provide limited constructs, which are not expressive enough to offer appropriate support. Some approaches, e.g. RMM [10] and OOHDM [15], only provide navigational constructs to allow the description of navigational structures; the composition of sophisticated user interfaces is either beyond their considerations or is addressed by external constructs or built-in constructs. Other approaches, like ARANEUS [1], WebML [3] and OntoWebber [11], do provide explicit constructs to model the composition of user interfaces. However, their constructs are typically defined at a coarse-grained level. For example, OntoWebber provides a set of *card primitives* and WebML defines a set of *unit primitives* to model typical content for web pages. They claim that *a card* or *a unit* is the minimal unit of a site view. However, each card or unit should consist of a number of components, e.g. text,

hyperlink and images. Therefore, more fine-grained concepts should be proposed to allow cards or units to be composed according to complex requirements.

The support for modelling presentation styles and layouts has not been addressed in most approaches. They typically rely on external approaches (e.g. style sheets) to facilitate the specification of presentation styles. Moreover, they do not provide any means to address the layout for web pages. This is because the declarative specification of the web pages is not available and components within the web pages are not addressable. As a consequence, web developers still need to do a lot of low-level work to create complex presentation styles and layouts for web pages.

5 Conclusions and Future Work

In this paper, we have illustrated the OntoWeaver site ontologies for modelling data-intensive web sites. OntoWeaver distinguishes itself from other web site modelling approaches in several ways. First, it employs ontology as the backbone to drive the design and management of data-intensive web sites. This approach enables web sites to be represented declaratively in an exchangeable format, and thus enables the high level support for the design and development. Second, it proposes a site view ontology to model the site views of data-intensive web sites at a fine-grained level. In this way, OntoWeaver offers high level support for the creation of sophisticated site views. Third, OntoWeaver proposes a presentation ontology, which models presentation styles and layouts of web pages. Hence, presentation styles and layouts can be specified at a high level of abstraction. Finally, OntoWeaver proposes a generic customization framework, which takes advantage of the declarative specifications of data-intensive web sites and offers comprehensive customization support.

A prototype system of OntoWeaver, including all the tools mentioned in this paper, has been implemented. Future work focuses on i) defining constraints validating the complex site specifications, ii) providing tools helping developers to find and correct the specifications that are either with errors or being inconsistent in the entire site model, and iii) using the emerging semantic web standard OWL [18] as the underlying language to represent web applications to ensure that the entire model of the target web application can be exploited by semantic-aware applications and thus improve the performances of the web applications on knowledge sharing, knowledge exchanging and personalization.

Acknowledgements

We would like to thank Maria Vargas-Vera for her insightful comments on earlier drafts of this paper.

References

1. P. Atzeni, G. Mecca, P. Merialdo, *Design and Maintenance of Data-Intensive Web Sites,* proceeding of the 6th int. Conference On Extending Database Technology (EDBT), Valencia, Spain, March 1998.
2. T. Berners-Lee, J. Hendler and O. Lassila, *The Semantic Web,* Scientific American, May 2001.
3. S. Ceri, P. Fratenali, A. Bongio. *Web Modelling Language (WebML): a modelling language for designing Web sites*. WWW9 Conference, Amsterdam, May 2000.
4. H. Eriksson, A. R. Puerta, and M. A. Musen (1994), *Generation of Knowledge-Acquisition Tools from Domain Ontologies*, Int. J. Human-Computer Studies (1994) 41, 425-453.
5. F. Frasincar, G. Houben, and R. Vdovjak, *Specification Framework for Engineering Adaptive Web Applications*, In the Eleventh International World Wide Web Conference WWW2002.
6. F. Garzotto, P. Paolini and D. Schwabe, *HDM—A Model-Based Approach to Hypertext Application design*, ACM Trans. Inf. Syst. 11, 1 (Jan. 1993), Pages 1 – 26.
7. W. E.Grosso, H. Eriksson, R. W. Fergerson, J. H. Gennari, S. W. Tu, and M. A. Musen (1999), *Knowledge Modelling at the Millennium*, In Proc. the 12th International Workshop on Knowledge Acquisition, Modelling and Management (KAW'99) Banff, Canada, October 1999.
8. T. R. Gruber, *Toward Principles for the Design of Ontologies Used for Knowledge Sharing*, In Formal Ontology in Conceptual Analysis and Knowledge Representation, edited by Nicola Guarino and Roberto Poli, Kluwer Academic Publishers, in press.
9. R. Hennicker and N. Koch. *A UML-based Methodology for Hypermedia Design*. In A. Evans, S. Stuart, and B. Selic, editors, Proc. of UML 2000 Conference, York, England, Oct. 2000. Springer LNCS 1939.
10. T. Isakowitz, E.A. Stohr and P. Balasubramaninan, *RMM: A Methodology for Structured Hypermedia Design*, Communications of the ACM, August 1995.
11. Y. Jin, S. Decker, Gio Wiederhold, *OntoWebber: Model-Driven Ontology-Based Web site Management*, Semantic Web Workshop, Stanford, California, July 2001.
12. Y. Lei, E. Motta and J. Domingue, *Design of Customized Web Applications with OntoWeaver*. In proceedings of the International Conference on Knowledge Capture, October, Florida, USA, 2003, pp 54-61.
13. 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.
14. E. Motta, S. Buckingham Shum, and J. Domingue (2000), *Ontology-Driven Document Enrichment: Principles, Tools and Applications*, Int. J. Human-Computer Studies (2000) 52, 1071-1109.
15. D. Schwabe and G. Rossi, *An Object Oriented Approach to Web-Based Application Design*, Theory and Practice of Object Systems 4(4), 1998. Wiley and Sons, New York, ISSN 1074-3224.
16. *Resource Description Framework (RDF) Model and Syntax*, W3C Proposed Recommendation. http://www.w3.org/TR/PR-rdf-syntax/.
17. *Resource Description Framework (RDF) Schema Specification 1.0*, W3C Candidate Recommendation, http://www.w3.org/TR/2000/CR-rdf-schema-20000327/, 2000.
18. O*WL Web Ontology Language*, W3C Working Draft, March 2003, http://www.w3.org/TR/2003/WD-owl-features-20030331/.