Journal of Web Engineering, Vol. 4, No.3 (2005) 244-262

ONTOWEAVER: AN NTOLOGY-BASED APPROACH TO THE DESIGN OF DATA-INTENSIVE WEB SITES

YUANGUI LEI, ENRICO MOTTA, JOHN DOMINGUE

Knowledge Media Institute, the Open University, Milton Keynes

{y.lei, e.motta, j.b.domingue}@open.ac.uk

Received October 31, 2004
Revised February 28, 2005

Building a data-intensive web site is a complex task. Ad hoc rapid prototyping approaches easily lead to unsatisfactory results, e.g. poor maintainability and extensibility. To address this problem, a number of model-based approaches have been proposed, which attempt to simplify the design and development of data-intensive web sites. However, these approaches typically lack expressive meta-models and, as a result, suffer from a number of limitations, e.g. the lack of appropriate support for the creation of complex user interfaces, for the specification of layouts and presentation styles, and for customization.

In this paper we describe a new software tool OntoWeaver, which uses ontologies to drive the design and development of data-intensive web sites. OntoWeaver overcomes the problems of current approaches by providing *a site view ontology*, *a presentation ontology*, and *a customization framework*. Specifically, the site view ontology provides fine-grained modelling support for the creation of complex user interfaces and navigation structures. The presentation ontology captures the features of layouts and presentation styles of user interface elements. These two explicit meta-models allow the target web site to be represented in a declarative and re-usable format, thus enabling high level support for design, maintenance, and customization. The customization framework exploits this advantage and provides comprehensive customization support for the target web site at design as well as run time.

Keywords: Web Modelling, Customization, Data-Intensive Web Site, Web Site Design, Ontology

1. Introduction

Building a data-intensive web site is a complex task. It involves not only technical issues, but also organizational, managerial and artistic issues [19]. Ad hoc rapid prototyping approaches easily lead to unsatisfactory results, e.g. poor maintainability and poor extensibility [21]. To address this problem, a number of model-based approaches have been proposed, which describe web applications at a conceptual level without committing to detailed implementation issues [8, 12, 23, 1, 5, 9, 4, 13, 6]. These approaches provide high level support for web site design from conceptualization and specification down to maintenance, by distinguishing different dimensions of web design, organizing the development activities into a well-structured process, providing models to facilitate the specification, and offering tools with varying levels of automation [7, 15, 22].

One problem common to all these approaches is the relatively little support for the composition of complex user interfaces. While some support is provided by existing tools to construct user interfaces, the resulting interfaces are rather limited as only simple primitives are provided. Furthermore, no fine-grained modelling support is available to allow web developers adapting such user interfaces, e.g. removing or adding specific web content.

Another problem is the lack of modelling support for the layouts and presentation styles of user interface elements. Web developers have to rely on ad hoc approaches, e.g. cascading style sheets (CSS) [27] and implementation level coding approaches, to achieve a specification. This becomes time-consuming when a web site needs to be rendered in different ways for different purposes, such as different devices, user groups, individuals etc.

In addition to the problems mentioned above, the lack of appropriate meta-models acts as a barrier to providing appropriate support for intelligent analysis and management of target web sites. For example, because not all aspects of web sites are represented declaratively and can be reasoned upon, high-level support for customizing and validating web site designs is limited. To address these problems, expressive meta-models are required, which are powerful enough to allow all aspects of data-intensive web sites to be represented declaratively. For this purpose, we have implemented OntoWeaver, an ontology-based approach that relies on a set of expressive meta-models to drive the design and development of data-intensive web sites [16, 17, 18].

In this paper we show how OntoWeaver facilitates the design and development of data-intensive web sites. We begin in section 2 by exploring the open issues associated with current approaches. We then clarify the requirements placed upon web site design frameworks in order to address these issues. In section 3 we present the design principles of the OntoWeaver approach. We then illustrate the major components of OntoWeaver in sections 4, 5, and 6. Finally, in section 7 we draw the main conclusions on our work and outline future work.

2. Web site design through conceptual modelling

In this section we briefly present an abstract architecture underlying current web modelling approaches. We then employ this architecture to explore the open issues associated with current approaches and clarify how these issues can be addressed.

2.1. An abstract architecture for user modelling

In the area of conceptual web modelling, a number of approaches have been derived from the Dexter Reference Model [11], focusing on the design and development of data-intensive web sites. They typically describe the architecture of data-intensive web sites as the composition of the following three layers [7, 22]:

- *A data layer*, which describes the underlying domain data of the target web site. It comprises three major components: *a domain data model*, which expresses the data structure of the problem domain, *databases*, which store data objects, and *a set of data services*, which allow access to, updating, and querying of the underlying domain data.
- *A navigation structure layer*, which describes the navigation structures of the target web site. The major components of this layer are *page nodes*, which denote web pages, and *links*, which describe link relations between web pages.
- *A user interface layer*, which describes the user interfaces of the target web site. It comprises elements that are able to present navigation structures, domain data, and forms that allow data accessing services, e.g. data acquisition and data querying.

This three-layer architecture separates the specification of the target web site from the underlying domain data. It relies on the navigation structure layer and the user interface layer to support access to domain data. Current approaches distinguish these three layers and provide models to address them accordingly. In particular, a number of comprehensive methods and primitives have been proposed to

support the design of the navigation structure layer. Examples include the *access primitives* (such as *index*, *guided tours*, *uni-directional links*, and *bi-directional links*) in the Relationship Management Methodology (RMM) [12]; the *navigational classes* in the Object-Oriented Hypermedia Design Method (OOHDM) [23]; the *navigational conceptual model* (NCM) in ARANEUS [1]; the *navigation model* in i) the UML-based Web Engineering approach (UWE) [15], ii) the Web Modelling Language (WebML) [4], and iii) OntoWebber [13]; and the *Navigation Access Diagram* (NAD) in OOH [9].

The user interface layer has been explicitly addressed in most approaches. For example, OOHDM relies on an external approach to describe the user interfaces of web applications. It maps each navigation object (e.g. *node* and *link*) to an abstract user interface object. ARANEUS proposes a logical data model called *ADM* to represent an abstract description of actual web pages. UWE also provides an abstract user interface model to support the design of user interfaces. Approaches like OOH, WebML, and OntoWebber propose comprehensive primitives to describe typical user interfaces of data-intensive web sites, such as the user interfaces for data presentation, data acquisition, and data querying.

As web sites offer information which is potentially interesting to a wide range of audiences, they are required to be capable of presenting customized views to individual users. This requirement has been addressed in current approaches by means of the following methods:

- *User group specific customization*, which customizes the target web site for individual users according to the user groups they fall in. WSDM [5] and OOH support this type of customization by using the requirements from the target users to drive the design and development of web sites. End users are classified into user groups. Navigation structures and user interfaces are then designed for each user group. One major problem of this methodology is that it only reflects customization requirements of user groups. Individual personalization requirements are not taken into consideration. Another problem is that it cannot scale up. As the number of user groups grows, the workload of designing and maintaining a large number of site models becomes too heavy.
- *User specific customization*, which derives customized views for individuals according to their profiles and customization requirements. Two major solutions have been developed to support this type of customization. One is the solution employed in approaches like the extended OOHDM approach [24], WebML, and HERA [6], which employs user information to annotate the specification of the target web site. The other solution is the one adopted in UWE and WUML [14], which relies on *a user model* to describe user profiles, a set of *customization rules* to specify customization requirements, and *an application model* to describe the target web site.

2.2. Open issues

As discussed above, current approaches distinguish different layers to describe data-intensive web sites and provide models to address each layer accordingly. In particular, they provide comprehensive support to facilitate the design of navigation structures. Furthermore, most approaches provide abstract user interface models to address the design of user interfaces explicitly. Finally, most approaches take customization into consideration and provide customization support for the target web site to varying levels. However, there are a number of open issues, which need to be addressed:

- *Relatively little support for the composition of user interfaces.* Although comprehensive coarse-grained primitives have been proposed in most approaches to model typical user interfaces of web pages, no further modelling support is available to allow the adaptation of such typical user interfaces. For example, no modelling support is provided to address *atomic* user interface elements, which are components of typical user interfaces of data-intensive web sites, such as elements that i)

present static information, ii) present dynamic information, iii) allow input from end users, and iv) allow the invocation of available services. As a consequence, web developers are only able to express a fixed number of typical user interfaces in terms of the provided primitives. The creation of complex user interface is out of reach at the conceptual level.

- *Little support for the specification of the layouts and presentation styles for user interface elements.* Most approaches do not take layouts and presentation styles of user interface elements into consideration. As a consequence, web developers therefore have to rely on ad hoc approaches, e.g. CSS, and low-level programming to define and maintain the specification. In particular, web developers have to use programming approaches to specify layouts for the target web site. OntoWebber is a partial exception, which proposes a set of layout primitives (e.g. *flow layout* and *grid layout)* to describe typical layouts of user interface elements. However, these primitive do not support the expression of complex layouts.
- *The lack of comprehensive customization support.* First, as discussed earlier, user group specific customization cannot provide a comprehensive customization support for the target web site, as it consider the customization requirements of individuals. Second, user specific customization support is limited in current approaches, as not all aspects of web sites are available for customization due to the lack of expressive meta-models for describing the target web site. Furthermore, most approaches, e.g. the extended OOHDM, WebML, HERA, and WUML do not separate the specification of customization from other aspects of data-intensive web sites. As a consequence, web developers have to anticipate what can be customized at the stage of navigation structure design and user interface design. Finally, specific support for the specification of individual customization requirements is not available. For example, in approaches like the extended OOHDM, WebML, and HERA, no specific models are available to support the specification of user annotation and the association of annotations with site specifications. Analogously, while approaches like UWE and WUML, which employ rules and user profiles to support customization, provide generic modelling support such as UML, no specific modelling support is available to allow, e.g., the definition of customization conditions and adaptation actions.

2.3. A new architecture

To address the issues discussed above, the three-layer architecture of current approaches needs to be extended. First, *a presentation layer* needs to be added on top of the user interface layer to emphasize the importance of the modelling support for layouts and presentation styles. Second, *a customization layer* needs to be added to separate customization requirements from other aspects of the target web site. This separation overcomes the problem caused by mixing the customization requirements with the specification of other aspects of the target web site. Figure 1 shows the extended architecture.

The *site view layer* is the combination of the navigation structure layer and the user interface layer. The combination avoids the overlapping between navigation elements and user interface elements[a]. In order to provide appropriate support for the specification of this layer, an expressive meta-model is crucial in web site design. Such meta-model should allow web developers creating complex site views according to their own requirements, rather than force them to shape their requirements to fit in the web site design approach. Therefore, an approach should be able to describe *static content*, which is defined at design time, *dynamic content*, which is retrieved from the underlying domain databases at

[a] Some elements in the site view layer can be seen as navigation elements as well as user interface elements. For example, page nodes in navigation structures can be seen as user interface elements, which in turn comprise a number of user interface components.

run time, and *the user interaction part*, which allows users to type information and interact with the target web site. Moreover, the meta-model should distinguish between *atomic* user interface elements and *composite* user interface elements and provide comprehensive constructs to address them accordingly, thus supporting user interface composition.

The *presentation layer* describes layouts and presentation styles for user interface elements. It separates layouts and presentation styles from user interface elements. Thus, different presentation instructions can be specified to support the generation of different presentations for the same site view. To allow presentation instructions to be specified at a high level without having to commit to implementation issues, an appropriate model is required, which captures features of this layer. In particular, the layouts of user interface elements should be addressed to allow high level support.

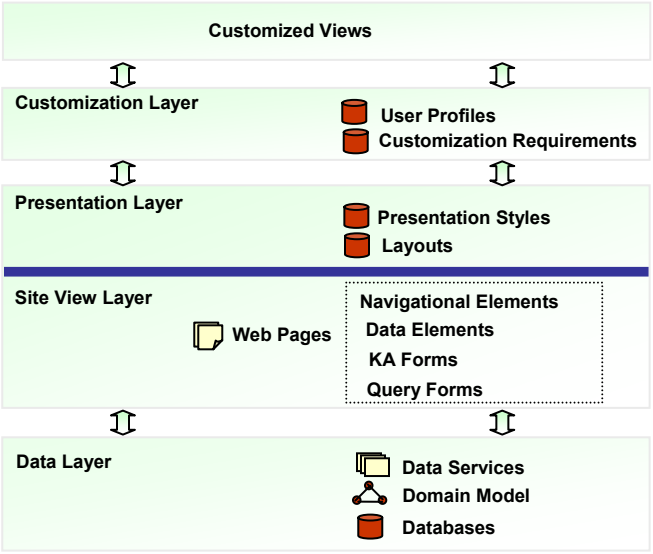

Figure 1 The extended architecture of data-intensive web sites.

The *customization layer* describes the customization requirements of web sites. The major components are user profiles and customization rules. User profiles store information about end users, such as preferences, environments etc. Customization rules specify the requirements of personalizing the target web site for individual users, e.g. the conditions that should be satisfied and the actions that personalize the presentation layer and the site view layer. To allow comprehensive customization support, all aspects of the target web site should be available for customization. Furthermore, an explicit customization framework should be provided to allow the high level specification of customization requirements at design time and to offer comprehensive customization support at run time.

3. An overview of the OntoWeaver framework

An ontology is an explicit formal specification of a conceptualization [10]. In the context of web site design, ontologies can be used to provide formal vocabularies for specifying the target web site in a declarative and re-usable format, thus enabling high level support for design and development. Based

upon this idea, we have implemented OntoWeaver, which uses ontologies as the backbone to drive the design and development of data-intensive web sites.

OntoWeaver addresses the open issues associated with current approaches by providing a set of explicit meta-models to capture features of each layer of data-intensive web sites. Specifically, it provides *a site view ontology*, *a presentation ontology*, and *a customization framework*. Figure 2 shows the architecture of the OntoWeaver framework. It accepts a domain ontology as input and produces a customized data-intensive web site for individual users.

The site view ontology models the site view layer of data-intensive web sites. As will be described in section 4, it provides comprehensive modelling support for typical user interface elements, generic composite user interface elements, as well as atomic user interface elements. It realizes a mechanism to support the composition of complex user interfaces.

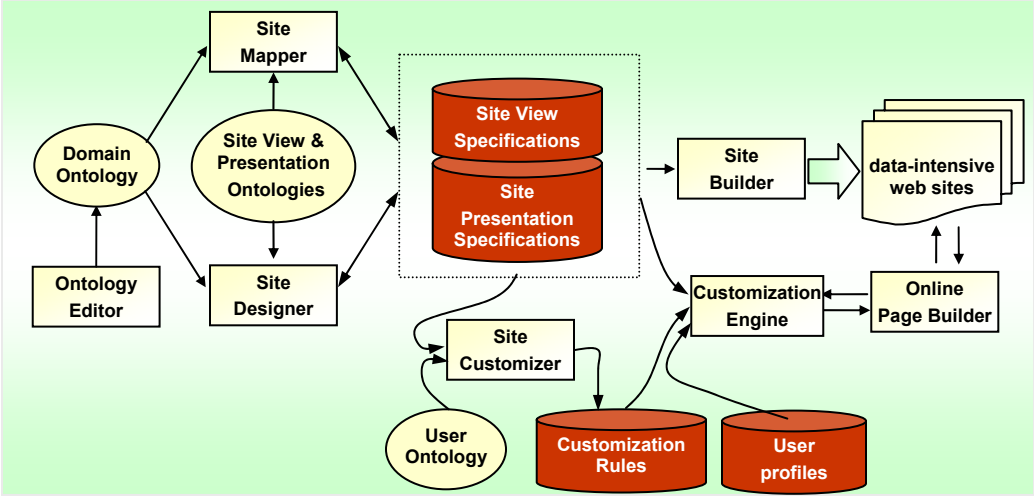

Figure 2 The architecture of the OntoWeaver framework.

The presentation ontology captures the features of the layouts and presentation styles of user interface elements. It addresses the second open issue discussed in section 2.2. On the one hand, the presentation ontology allows web developers to specify the layouts and presentation styles of user interface elements at the conceptual level. On the other hand, it enables the specification of layouts and presentation styles to be represented separately from user interface elements. Hence, different rendering styles of the target web site can be easily specified and maintained for different purposes.

The site view ontology and the presentation ontology allow the target web site to be represented declaratively. In particular, all user interface elements and their presentation instructions are described declaratively and as a result available for customization. OntoWeaver exploits this advantage and proposes a customization framework, which enables comprehensive customization support at design as well as run time. Specifically, as will be discussed in section 6, the customization framework separates the specification of customization from other aspects of the target web site. It proposes a customization rule model to provide specific high level support for the specification of customization rules at design time. Furthermore, the customization framework enables comprehensive customization support at run time, by applying customization rules to reason upon user group specific site specifications to derive customized views for user individuals according to their profiles.

A typical design process in OntoWeaver proceeds by iterating the following steps: i) designing the domain ontology; ii) specifying navigation structures and composing user interfaces; iii) defining layouts and presentation styles, and iv) expressing customization requirements. As shown in figure 2, OntoWeaver provides a set of tools to support the design activities and the generation of customized data-intensive web sites. Specifically, the *Ontology Editor* allows developers to create and edit the ontologies associated with the target web site. The *Site Designer* supports the design tasks needed for specifying a data-intensive web site. The *Site Mapper* produces a default specification for the target web site and is responsible for re-engineering the web site specification after the domain ontology has been modified. The *Site Builder* validates the site specifications and compiles site specifications into web site implementations. The *Site Customizer* supports the activity of specifying customization requirements. The *Customization Engine* performs inferences upon the site specifications. The *Online Page Builder* generates customized web pages on the fly from the customized site specifications produced by the customization engine.

The semantic web [2] is a vision of the next generation of the current Web in which information is given well-defined meaning understandable to machines as well as to human beings. In order to allow the target web site to fit in this vision, OntoWeaver uses the emerging semantic web standard – the Resource Description Framework (RDF) [25] and RDF Schema [26] to represent its specification, thus allowing the target web site to be able to be picked up by semantic-aware web applications. However, as these languages are not powerful enough to describe the constraints and relationships among ontologies, the internal knowledge model of OntoWeaver is frame-based and compatible with OCML [20].

Figure 3 shows the main classes of the domain ontology abstracting the institutional information of our research department, the Knowledge Media Institute (KMi) at the Open University. The KMi web portal allows general users to browse and query the underlying domain data and allows advanced users to add new data entries. We will use this example to illustrate the main components of OntoWeaver throughout the rest of the paper.

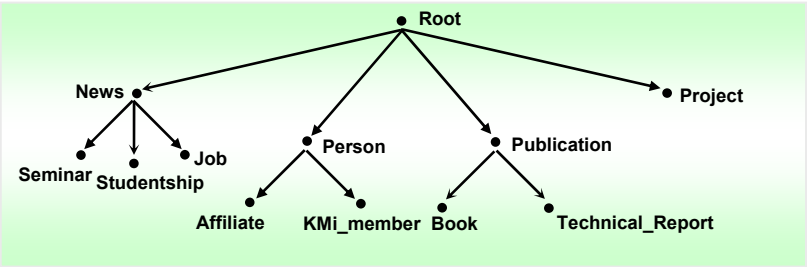

Figure 3 The main classes of a domain ontology of the KMi web portal.

4. The site view ontology

The site view ontology addresses the site view layer according to the requirements elaborated in section 2.3. It provides a set of comprehensive *navigation constructs* to support the specification of navigation structures. Furthermore, unlike current approaches, which do not address basic user interface elements, the site view ontology distinguishes between atomic user interface elements and composition user interface elements and provides comprehensive constructs to address them accordingly, thus realizing a mechanism to support the composition of user interfaces.

4.1. Navigation constructs

Links play a crucial role in web sites as they are the major components supporting navigation. To specify links, URLs of the associated linked web pages are required, which can be pre-defined (i.e. *static links*) or retrieved from the underlying domain data layer (i.e. *dynamic links*). In the case of *contextual links*, contextual information needs to be specified to ensure correct information flow from the source pages to the linked web pages. The site view ontology distinguishes these three types of links and provides the following constructs to describe them:

- The construct *LinkItem*, which abstracts static links in terms of *hasAssociatedResourceURI* specifying the URL of the linked web resource.
- The construct *DynamicLinkItem*, which relies on slots *hasClassEntity* and *hasSlotEntity* to specify the source of the URL of the linked web resource. The following RDF code[b] defines a dynamic link in which the URL of the linked web resources comes from the slot *has-web-address* of the class *Project* (The prefix 'svo' refers to the namespace of the site view ontology: xmlns:svo=http://kmi.open.ac.uk/people/yuangui/siteviewontology#. The prefix 'do' refers to the namespace of the underlying domain ontology of the target web site).

```
<rdf:Description rdf:about="…/project-url-link" >
  <rdf:type rdf:resource="&svo;DynamicLinkItem"/>
  <svo:hasClassEntity rdf:resource="&do;Project" />
  <svo:hasSlotEntity rdf:resource="&do;has-web-address" />
</rdf:Description>
```

- The construct *ContextualLinkItem*, which relies on a slot called *hasInstanceConstraint* to describe the associated contextual information constraining the data content presented in the linked web page. A contextual link example in the KMi web portal is the link in the project web page, which allows navigation to the web page to present the detailed information about the specified person. This link can be associated with different user interface elements, which present the names of people relevant to each project instance, e.g. *leaders* or *members*. The following code illustrates the specification of this contextual link. The contextual information constrains the instances of the class *Person*, using the person name that an end user clicks on as the filter of the slot *has-name*. The filtering value is specified as "parent.value", which means the value of the *output* element which visualizes the contextual link. As will be described in the following section, links rely on output elements to realize their visualization.

```
<rdf:Description rdf:about="…/project-member-contextual-link" >
  <rdf:type rdf:resource="&svo;ContextualLinkItem" />
  <svo:hasAssociatedResourceURI>person_page.jsp</svo:hasAssociatedResourceURI>
  <svo:hasInstanceConstraint rdf:resource="#person-name-constraint" />
</rdf:Description>

<rdf:Description rdf:about="…/person-name-constraint" >
  <rdf:type rdf:resource="&svo;InstanceConstraint" />
  <svo:hasConstrainedClassEntity rdf:resource="&do;Person" />
  <svo:hasConstrainedSlotEntity rdf:resource="&do;has-name" />
  <svo:hasConstrainedRelation  rdf:resource="#EQUAL"/>
  <svo:hasConstrainedValue>parent.value</svo:hasConstrainedValue>
</rdf:Description>
```

4.2. The atomic user interface constructs

OntoWeaver distinguishes three types of atomic user interface elements. They are *output elements*, which present static or dynamic information pieces, *input elements*, which allow end users to input information, and *command elements*, which allow end users to invoke the associated services. Input

[b] To enable readability, the URIs of the entities illustrated in this paper are simplified.

elements and command elements are typically used in knowledge acquisition and data querying forms. OntoWeaver provides the following constructs to address these basic elements:

- The construct *Output*, which models the output elements that present *static* information. It relies on slots *hasOutputValueType* and *hasOutputValue* to specify the presented information, which can be text or image. The presented information can be associated with links. Such association is described by means of the slot *hasLinkItem*.
- The construct *DynamicOutput*, which expresses those output elements that present values of the specified slot of the associated class entity. It is a sub-class of the construct *Output*. It employs slots *hasClassEntity* and *hasSlotEntity* to indicate the source of the dynamic data that will be presented.
- The construct *Input*, which describes input fields. Like *DynamicOutput*, this construct employs slots *hasClassEntity* and *hasSlotEntity* to specify the concept that the information gathered from the input element corresponds to.
- The construct *Command*, which abstracts command elements by means of slots *hasTask* and *hasAssociatedResourceURI*. The slot *hasTask* defines the associated task. OntoWeaver provides a set of built-in services for target web sites, such as data retrieving, data querying, and data acquisition to allow the access of the data layer. The slot *hasAssociatedResourceURI* specifies the web resource, which will be presented after the invocation of the associated task. This associated web resource typically presents the results of the associated task.

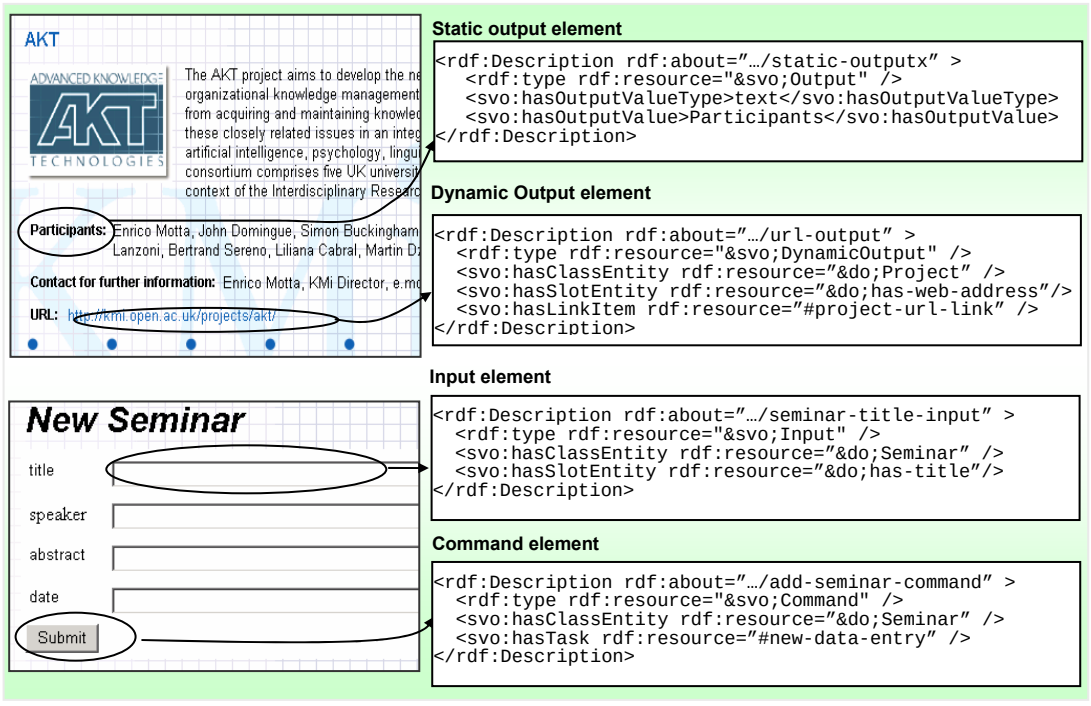

Figure 4 Examples of the atomic user interface elements and their RDF definitions.

Figure 4 illustrates some examples of atomic user interface elements. Specifically, the dynamic output element example presents values of the slot *has-web-address* for instances of the class *Project*. It is associated with a dynamic link, in which the URL of the linked web page comes from the value of the slot *has-web-address*. This link has been illustrated in the section above. The input element allows

end users typing information for the slot *has-title* to enable information gathering for a new data entry for the class *Seminar*. The command element is associated with one of the pre-defined tasks, the task *new-data-entry*.

4.3. The composite user interface constructs

The composite user interface constructs model those site view elements, which are composed of a number of sub elements. They include:

- The construct *Site*, which models a web site as a composition of web pages.
- The construct *SiteResource*, which models web pages as compositions of components.
- The construct *ResourceComponent*, which models content of web pages as compositions of atomic user interface elements and sub components.
- A set of *component primitives*, which model typical dynamic user interfaces of web pages. OntoWeaver distinguishes three kinds of typical user interfaces in data-intensive web sites, which are user interfaces for data presentation, data acquisition, and data querying. OntoWeaver provides constructs *DataComponent*, *KAComponent*, and *SearchComponent* to address them accordingly. Specifically, the construct *DataComponent* describes user interface elements, which present instances of the specified domain class. It relies on the slot *hasClassEntity* to specify the domain class and the slot *hasInstanceConstraint* to specify constraints to filter instances of the associated class. The construct *KAComponent* models user interface elements, which allow the acquisition of data facts from end users for the specified domain class. The construct *SearchComponent* abstracts user interface elements, which allow the querying of the domain data. Both of these two latter constructs rely on the slot *hasClassEntity* to indicate the associated domain class.

4.4. User interface composition

Figure 5 shows an overview of the site view ontology. It relies on the composite constructs, e.g. *SiteResource* and *ResourceComponent*, and *the atomic user interface constructs* to realize a mechanism, which allows the composition of complex user interfaces. Specifically, the user interfaces of web pages are composed of a number of resource components. Each resource component further contains atomic user interface elements and sub resource components. Thus, a complex user interface can be composed.

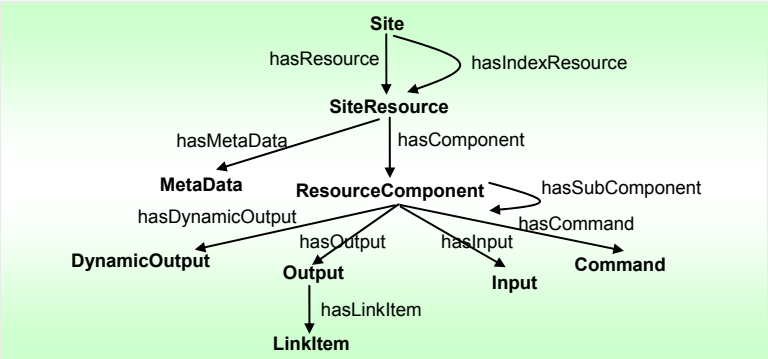

Figure 5 An overview of the site view ontology.

Adapting typical user interfaces of data-intensive web sites. As discussed earlier in section 2, current approaches do not support the adaptation of typical user interfaces, due to the lack of

expressive user interface models. Now we investigate how OntoWeaver addresses this problem. We use the user interface of data components as an example. As shown in part (a) of figure 6, the default user interface of a data component is composed of a number of dynamic output elements presenting values of slots for instances of the specified class and a number of static output elements presenting explanations about the dynamic values. Each sub element is specified declaratively and available for modification. Furthermore, new elements can be easily added into the user interface, as the construct *DataComponent* supports the flexible assembling of user interface elements. Part (b) of figure 6 shows an adapted user interface example: the static output elements for presenting explanations about the values of slots *project_name*, *picture*, and *description* have been removed, as their explanations can be indicated by their content or presentation styles. The output type of the dynamic output *picture* has been changed from text to image.

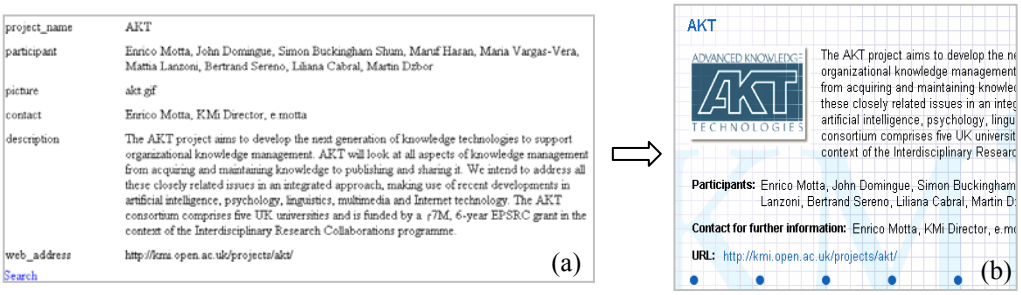

Figure 6 An example of adapting typical user interfaces. Part (a) shows the default user interface of the project data component. Part (b) shows the adapted user interface.

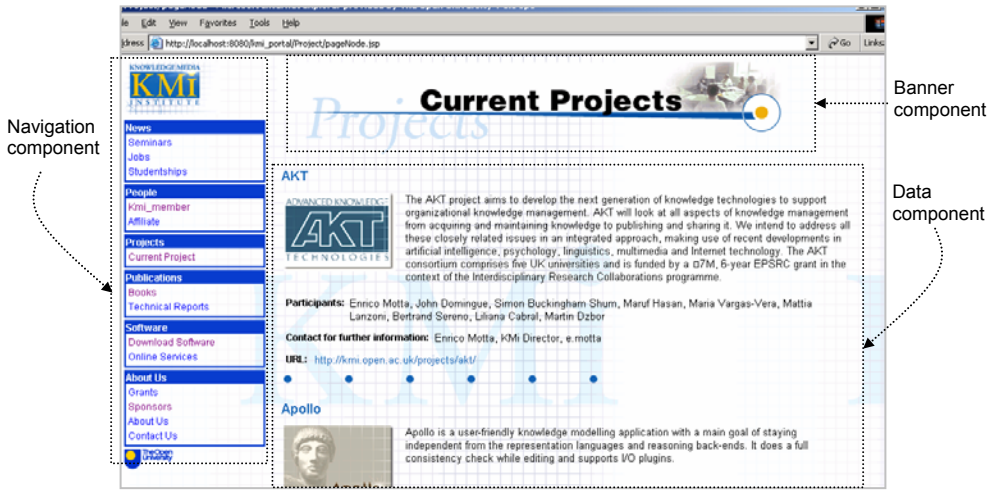

Figure 7 A sample user interface for a web page presenting projects in the KMi Web Portal.

Composing user interfaces for web pages. Figure 7 shows a sample user interface of a web page, which is composed of three components: *a navigation component* presenting hyperlinks, *a banner component* displaying a banner for the web page, and *a data component* presenting instances of the class *Project*. Each component is further made up of a number of sub-elements. For example, the navigation component is composed of a number of sub components. Each sub component further contains a number of output elements. Current approaches would specify such a user interface only by means of a fixed number of primitives. Mechanisms to support composition are not provided.

OntoWeaver on the other hand allows the specification of atomic user interface elements and also supports the assembling of user interface elements. The following code illustrates the composition of the sample user interface.

```
  <!-- the user interface of the web page is composed of a set of components -->
<rdf:Description rdf:about="…/project-page" >
  <rdf:type rdf:resource="&svo;SiteResource" />
  <svo:hasComponent>
    <rdf:Bag>
       <rdf:li rdf:resource="…/navigationcomponent"/>
       <rdf:li rdf:resource="…/bannercomponent"/>
       <rdf:li rdf:resource="…/datacomponent"/>
    </rdf:Bag>
  </svo:hasComponent>
</rdf:Description>
  <!-- the navigation component comprises a number of sub components -->
<rdf:Description rdf:about="…/navigationcomponent" >
  <rdf:type rdf:resource="&svo;ResourceComponent" />
  <svo:hasComponent>
     <rdf:Bag>
       <rdf:li rdf:resource="…/Newscomponent"/>
       …
       <rdf:li rdf:resource="…/Aboutuscomponent"/>
    </rdf:Bag>
  </svo:hasComponent>
</rdf:Description>
…
  <!-- the news component is composed of a number of output elements -->
<rdf:Description rdf:about="…/Newscomponent" >
  <rdf:type rdf:resource="&svo;ResourceComponent" />
  <svo:hasOutput>
       …
  </svo:hasOutput>
</rdf:Description>
…
```

5. The presentation ontology

The presentation ontology provides explicit vocabularies to allow the specification of presentation instructions for the target web site. It addresses both layouts and presentation styles. As shown in figure 8, the presentation ontology describes a presentation instruction of the target web site as a collection of *templates*, *presentation objects*, and *layout objects*. Templates describe presentation styles e.g. backgrounds, colours, and fonts. Presentation objects specify templates for user interface elements. Layout objects express organization instructions. The reference of user interface elements in the presentation model is realized by means of their Uniform Resource Identifiers (URIs) [3].

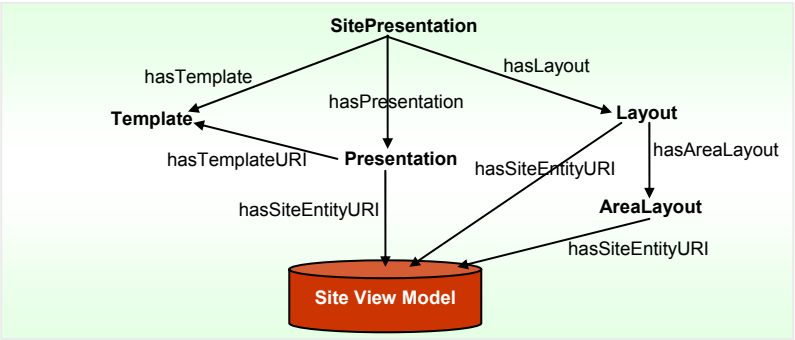

Figure 8 An overview of the presentation ontology.

There are three template constructs proposed in the presentation ontology. They are *GenericPresentationTemplate*, which describes generic presentation styles shared in most user interface elements, e.g. background, colours, and fonts, *WidgetPresentationTemplate*, which describes presentation styles for user interface elements where widgets are involved, e.g. dynamic output

elements, input elements, and command elements, and *DataComponentPresentationTemplate*, which works on data components presenting dynamic data content.

Two layout constructs are provided to model organizations of site view elements: i) *TextLayout ,* which models the layout of atomic user interface elements in terms of *alignment*, specifying the alignment of a user interface element within the component that contains this element, and ii) *ComponentLayout*, which abstracts the organization features of composite interface elements. Specifically, a component layout places the sub-elements of the specified component into five sub areas, which are *top*, *left*, *middle*, *right*, and *bottom*. Each area can display a number of elements in a specified layout direction, i.e. horizontal or vertical. OntoWeaver relies on the construct *AreaLayout* to describe such organization of each area.

Now we investigate how the presentation ontology facilitates the specification of presentation instructions for the target web site. We use the sample user interface shown in figure 7 as a study case. A number of templates have been defined for rendering the user interface elements. For example, a template has been defined for the dynamic output element that presents values of the slot *has-title*, which renders the values of project titles in a bold font with a slightly large size with no widget involved. The following code illustrates how to specify such a template for a user interface element.

```
<rdf:Description rdf:about="…/template1" >
  <rdf:type rdf:resource="&spo;WidgetPresentationTemplate" />
  <spo:hasWidgetType>None</spo:hasWidgetType>
  <spo:hasFontBold>True</spo:hasFontBold>
  …
</rdf:Description>

<rdf:Description rdf:about="…/presentation1" >
  <rdf:type rdf:resource="&spo;Presentation" />
  <spo:hasTemplate rdf:resource="…/template1" />
  <spo:hasSiteEntityURI>…/dynamic-title</spo:hasSiteEntityURI>
</rdf:Description>
```

Regarding the layout of the sample user interface, the navigation component is placed in the left area. The banner component and the data component are arranged in the middle area. For each component, a layout is further specified according to different requirements. For example, the layout of the data component arranges the sub component *project_name* at the top, *picture* at the left, *description* at the middle, and other sub-components at the bottom. These layout designs can be easily specified by means of the OntoWeaver layout constructs. The following fragment of RDF code[c] illustrates the layout specification of the data component, which only arranges the top-level sub-components. As each sub component can have its own layout design, a complex layout can therefore be specified for a user interface element.

```
<rdf:Description rdf:about="…/componentlayout1" >
  <rdf:type rdf:resource="&spo;ComponentLayout" />
  <spo:hasSiteEntityURI>…/datacomponent</spo:hasSiteEntityURI>
  <spo:hasTopAreaLayout rdf:resource="#componentlayout1_toparea"/>
  …
  <spo:hasBottomAreaLayout rdf:resource="#componentlayout1_bottomarea"/>
</rdf:Description>
…
<rdf:Description rdf:about="…/componentlayout1_bottomarea" >
  <rdf:type rdf:resource="&spo;AreaLayout"/>
  <spo:hasSiteEntityURI>
    <rdf:Bag>
       <rdf:li>…/datacomponent/participant</rdf:li>
       <rdf:li>…/datacomponent/contact</rdf:li>
       …
    </rdf:Bag>
  </spo:hasSiteEntityURI>
</rdf:Description>
```

[c] The prefix 'spo' refers to the namespace of the site presentation ontology: xmlns:spo=http://kmi.open.ac.uk/people/yuangui/sitepresentationontology#

6. The customization framework

As discussed earlier, current approaches lack comprehensive customization support for the target web site. First, not all aspects of the target web site are available to customization. This is addressed in OntoWeaver by means of the site view ontology and the presentation ontology. They allow all aspects of the navigation structures and user interfaces of the target web site to be represented declaratively. Second, most approaches, e.g. the extended OOHDM, WebML, HERA, and WUML do not separate the specification of customization from other aspects of data-intensive web sites. Web developers have to anticipate what can be customized in the stage of site view design. Finally, no specific support is available in current approaches to facilitate web developers specifying customization requirements for individual users. These problems are addressed in OntoWeaver by its customization framework.

As shown in figure 9, the customization framework relies on *customization rules* to describe when and how to perform certain customization actions, *user profiles* to capture user information, and a set of *user group specific site models* to describe the target web site. It references the site view elements in the specification of customization requirements by means of their uniform resource identifiers, thus realizing the separation of customization specification from other aspects of the target web site. *An inference engine* is employed, which applies rules to reason upon user group specific site specifications to derive customized views for individuals according to their profiles.

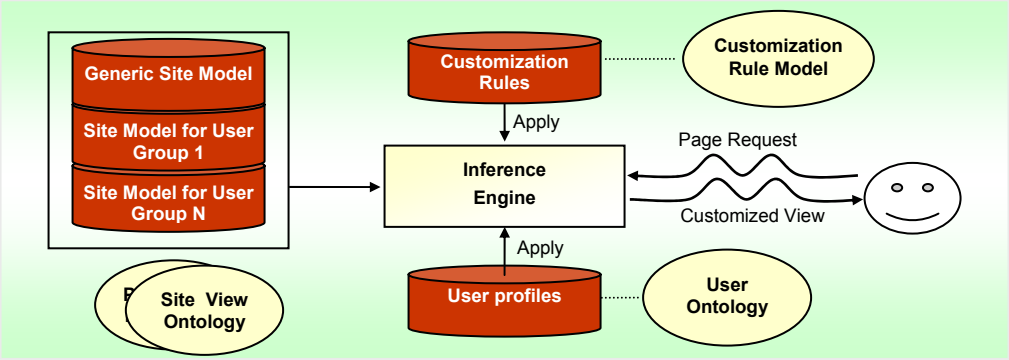

Figure 9 The OntoWeaver customization framework.

The customization framework proposes *a customization rule model* and *a generic user ontology*, which enable specific support for the specification of customization rules. In particular, the customization rule model provides comprehensive constructs to allow the formulation of customization conditions and the specification of adaptation actions. These are described below.

6.1. The user ontology

User information plays a crucial role in user specific customization. It is used to assess whether certain customization actions should take place or not. OntoWeaver provides a generic user ontology to describe general information about end users. The generic user ontology comprises two main classes: the class *User*, which describes user information in terms of *hasUserID*, *hasPassword*, *hasUserGroup*, *hasDevice*, and *hasInterest*, and the class *UserGroup*, which relies on slots *hasSiteViewURL* and *hasSitePresentationURL* to specify the user group specific site view model and the presentation model of the target web site. This ontology can be easily extended to abstract user information in the context of the problem domain.

6.2. The customization rule model

Customization rules define when and how to perform certain customization actions in terms of conditions and actions. The condition part describes a condition that has to be satisfied for the associated customization actions to take place. The action part describes the adaptation actions, e.g. adding/hiding/modifying components, or setting presentation or layout properties for components. To provide specific support for the construction of customization rules, OntoWeaver proposes a customization rule model. Figure 10 shows an overview of this model.

A customization condition can be atomic, which comprises only one condition, or composite, which is composed of a list of conditions by means of logical operators such as *AND*, *OR*, or *NOT*. Each condition is formulated by means of the construct *Condition*, which relies on slots *hasClassEntity* and *hasSlotEntity* to specify the user data which are going to be evaluated, and slots *hasRelationOperator* and *hasSpecifiedValue* to define the way to evaluate the condition, and the slot *hasLogicOperator* to specify how to connect this condition with the next one.

A customization action typically comprises three components: the slot *hasSiteEntityURI*, which indicates the site view element that the action works on, the slot *hasCustomizationType*, which specifies the customization type (e.g. site view, presentation, and layout), and the slot *hasModification*, which expresses customization details. Specifically, the slot *hasModification* describes how to customize the intended customization object. It relies on the class *Modification* to describe the customization details in terms of *slotName-value* pairs. The slot *slotName* indicates the slot of the object that customization intends to work on, while *value* specifies the customized value.

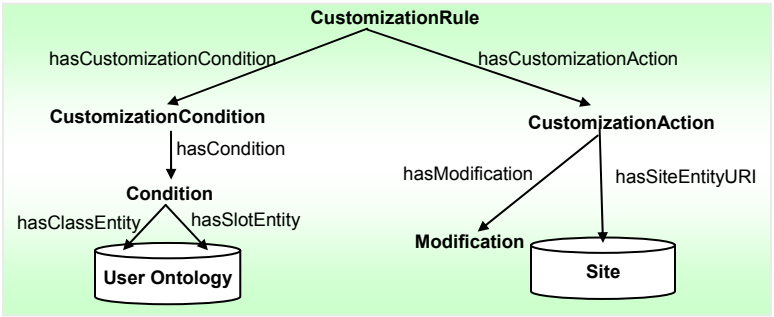

Figure 10 An overview of the customization rule model.

To illustrate how this rule model facilitates the specification of customization requirements of individuals, we investigate an example, which customizes the project web page, presenting only those projects that match the interest of end users. This customization requirement can be specified by a customization rule. The condition part specifies that the value of the slot *hasInsterest* of an end user should not be empty. The action part specifies constraints for the project data component. Specifically, it uses the interest of end users as constraints of the slot *addresses-research-area* to filter instances of the class *Project*. The following fragment of RDF code[d] defines this customization rule.

```
<rdf:Description rdf:about="…/customization-rule1" >
  <rdf:type rdf:resource="&sco;CustomizationRule" />
  <sco:hasCustomizationCondition rdf:resource="…/condition1" />
  <sco:hasCustomizationAction rdf:resource="…/action1" />
```

[d] The prefix 'sco' refers to the namespace of the customization rule model: xmlns:sco=http://kmi.open.ac.uk/people/yuangui/sitecustomizationontology#. The prefix 'uo' refers to the name space of the user ontology.

```
</rdf:Description>

<rdf:Description rdf:about="…/condition1" >
  …
  <sco:hasCondition rdf:resource="…/project-condition" />
</rdf:Description>

<rdf:Description rdf:about="…/project-condition" >
  <rdf:type rdf:resource="&sco;Condition" />
  <sco:hasClassEntity rdf:resource="&uo;User" />
  <sco:hasSlotEntity rdf:resource="&uo;hasInterest" />
  <sco:hasRelationOperator rdf:resource="#NOT" />
  <sco:hasSpecifiedValue>NULL</sco:hasSpecifiedValue>
</rdf:Description>

<rdf:Description rdf:about="…/action1" >
  <rdf:type rdf:resource="&sco;CustomizationAction" />
  <sco:hasSiteEntityURI>…/datacomponent/instanceFilter</sco:hasSiteEntityURI>
  <sco:hasCustomizationType rdf:resource="#SiteView" />
  <sco:hasModification rdf:resource="…/datacomponent-adaptation1" />
  <sco:hasModification rdf:resource="…/datacomponent-adaptation2" />
  …
</rdf:Description>

<rdf:Description rdf:about="…/datacomponent-adaptation1" >
  <rdf:type rdf:resource="&sco;Modification" />
  <sco:hasSlotEntity rdf:resource="&svo;hasConstrainedSlotEntity" />
  <sco:hasNewValue>"&do;addresses-research-area"</sco:hasNewValue>
</rdf:Description>

<rdf:Description rdf:about="…/datacomponent-adaptation2" >
  <rdf:type rdf:resource="&sco;Modification" />
  <sco:hasSlotEntity rdf:resource="&svo;hasConstrainedValue" />
  <sco:hasNewValue>"&uo;hasInterest"</sco:hasNewValue>
</rdf:Description>
…
```

6.3. Other components

The *declarative site models* are constructed for different user groups. They provide grounds for the rule-based customization to be carried out. OntoWeaver employs the Jess rule engine[e] to perform inferences. To this purpose, OntoWeaver provides an *RDF -> Jess* tool to convert the meta-models, site specifications and customization rules to Jess *templates*, *facts*, and *rules*.

The customization process starts when an end user logs into the OntoWeaver-generated web site and makes a page request. The tool *Online Page Builder* receives the page request, invokes the customization engine to perform inferences; gets the inference result from the customization engine; and builds s customized web page.

7. Conclusions and future work

In this paper we have presented OntoWeaver, a web site design framework, which uses ontologies to drive the design and development of data-intensive web sites. Specifically, we have identified a number of limitations exhibited by current web modelling approaches and illustrated how they are addressed in OntoWeaver by means of its three major components: the site view ontology, the presentation ontology and the customization framework.

The site view ontology provides fine-grained modelling support for user interfaces and navigation structures of the target web site. Unlike current approaches, which only address typical user interface elements of web pages, the site view ontology addresses the specification of atomic user interface elements, generic composite user interface elements, as well as typical user interface elements. It realizes a composition mechanism and allows web developers to express complex user interfaces according to their own requirements.

---

[e] http://herzberg.ca.sandia.gov/jess/index.shtml

The presentation ontology provides high level support for the specification of layouts and presentation styles for user interface elements. In particular, it allows web developers expressing complex layouts at the conceptual level. Web developers no longer need to encode the specification into web page implementations, like they have to do in other approaches. The extensible stylesheet language (XSL) [28] is a close approach, which provides comprehensive means to address the specification of presentation instructions. However, XSL exclusively focuses on the presentation of the source data stored in the specified XML document. Hence, it is very different from the presentation ontology.

OntoWeaver provides comprehensive customization support at design as well as run time. First, as all user interface elements and their presentation instructions are represented declaratively, the entire site model is available to customization. Second, OntoWeaver relies on its customization framework to separate the specification of customization from other aspects of the target web site, thus enabling the web site design process to be more flexible. Web developers do not need to anticipate what can be customized at the stage of site view design. Furthermore, OntoWeaver provides specific support for the specification of customization requirements. It offers a customization rule model to support the construction of customization rules. Finally, the customization framework takes advantage of both the rule-based customization approach and the user group specific customization approach to enable comprehensive customization support at run time.

Web site design critiquing is an important functionality for web site design frameworks. It allows developers to gain feedback and recommendations over the design result and helps developers to improve their design of the target web sites. At the moment, simple rules have been embedded within the OntoWeaver tools to support this functionality. In future, more powerful critiquing facility will be provided by i) defining complex constraints to verify the validity of complex site specifications and ii) allowing the specification of critiquing rules, thus offering customized critiquing service for web developers according to the their particular requirements.

We also plan to extend the customization framework and exploit a number of customization and adaptive techniques to provide a more comprehensive customization facility for the target web site. In particular, the issue of data integration will be investigated in the future in order to allow the re-use of user profiles which come from different customization technologies.

The semantic web is a vision of the next generation of the World Wide Web. How to design web sites which fit in this vision is a challenge for web site design frameworks. OntoWeaver can be seen as an initial approach to the design and development of such web sites, as it already employs semantic web technologies to benefit the web site design process. In future, more work will be done to ensure that OntoWeaver could make full use of the emerging semantic web, for instance by providing support for associating semantic mark-up with web pages at design time and for defining user interface components supporting semantic navigation.

Acknowledgements

We wish to thank Dr. Trevor Collins for his valuable comments on earlier drafts of this paper. 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. Atzeni, P., Mecca, G. and Merialdo, P., Design and maintenance of data-intensive web sites, In proceeding of the 6th International Conference on Extending Database Technology (EDBT), Valencia, Spain, March 1998.
2. Berners-Lee, T., Hendler, J. and Lassila, O., The Semantic Web. Scientific American, May, 2001.
3. Berners-Lee, T., Fielding, R. and Masinter, L., Uniform Resource Identifiers (URI): Generic Syntax, available online at: http://www.ietf.org/rfc/rfc2396.txt.4.
4. Ceri, S., Fratenali, P. and Bongio, A., Web Modelling Language (WebML): a modelling language for designing Web sites. WWW9 Conference, Amsterdam, May, 2000.
5. De Troyer, O. and Leune, C., WSDM: a user centered design method for Web sites, in proceedings of the Seventh International World Wide Web Conference, 1998.
6. Frasincar, F., Houben, G. and Vdovjak, R., Specification Framework for Engineering Adaptive Web Applications, In the Eleventh International World Wide Web Conference WWW2002.
7. Fraternali, P., Tools and approaches for developing data-intensive web applications: a survey. ACM Computing Surveys, Sept. 1999.
8. Garzotto, F., Paolini, P. and Schwabe, D., HDM—A Model-Based Approach to Hypertext Application design , ACM Trans. Inf. Syst. 11, 1 (Jan. 1993), Pages 1 – 26.
9. Gomez, J., Cachero, C. and Pastor, O., Conceptual modeling of device-independent Web applications. O. IEEE Multimedia , Volume: 8 Issue: 2 , April-June 2001. Page(s): 26 -39.
10. Gruber, T. R., 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.
11. Halasz, F. and Schewartz, M., The Dexter Hypertext Reference Model, CACM 37/2, Feb. 1994, pp.30-39.
12. Isakowitz, T., Stohr, E.A. and Balasubramaninan, P., RMM: A Methodology for Structured Hypermedia Design, Communications of the ACM, August 1995.
13. Jin, Y., Decker, S. and Wiederhold, G., OntoWebber: Model-Driven Ontology-Based Web site Management, Semantic Web Workshop, Stanford, California, July 2001.
14. Kappel, G., Retschitzegger, W., Pöll, B. and Schwinger, W., Modelling Ubiquitous Web Applications - The WUML Approach, In Proceedings of the International Workshop on Data Semantics in Web Information Systems (DASWIS 2001), Yokohama, Japan, November 2001.
15. Koch, N., Software Engineering for Adaptive Hypermedia Applications. PhD thesis, Reihe Softwaretechnik 12, Uni-Druck Publishing Company, Munich, 2001
16. Lei, Y., Motta, E. and Domingue, J., 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.
17. Lei, Y., Motta, E. and Domingue, J., Design of Customized Web Applications with OntoWeaver, In proceedings of the International Conference on Knowledge Capture, October, Florida, USA, 2003, pp 54-61.
18. Lei, Y., Motta, E. and Domingue, J., Modelling Data-Intensive Web Sites with OntoWeaver, In proceedings of the International Workshop on Web Information Systems Modelling (WISM 2004), Riga, Latvia, 2004, pp. 106-121.
19. Morville, P. and Rosenfeld, L., Information Architecture for the World Wide Web, O'Reilly, ISBN 1-56592-282-4, 1998.
20. Motta, E., Reusable Components of Knowledge Modelling: Case Studies in Parametric Design Problem Solving, IOS Press, Amsterdam, 1999.
21. Murugesan, S., Desshpande, Y., Hansen, S. and Ginige, A., Web Engineering: A New Discipline for Development of Web-based Systems, Web Engineering 2001, Page(s): 3-13.
22. Retschitzegger, W. and Schwinger, W., 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.

23. Schwabe, D. and Rossi, G., 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).
24. Schwabe, D.; Mattos Guimaraes, R.; Rossi, G., Cohesive design of personalized Web applications,. IEEE Internet Computing, Volume: 6 Issue: 2, March-April 2002, pp. 34 -43.
25. W3C, Resource Description Framework (RDF) Model and Syntax, W3C Proposed Recommendation, available online at http://www.w3.org/TR/PR-rdf-syntax/.
26. W3C, Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation, available online at http://www.w3.org/TR/rdf-schema/.
27. W3C, Cascading Style Sheets, available online at http://www.w3.org/Style/CSS/.
28. W3C, Extensible Stylesheet Language (XSL) Version 1.0, W3C recommendation, available online at http://www.w3.org/TR/xsl/LNCS 1392, 1998. 12-19