Abstract

In this article we present a method for the design and implementation of Web Applications for the Semantic Web. Based on the "Object Oriented Hypermedia Design Method" approach, we used ontology concepts to define an application conceptual model, extending the expressive power of the original method. The navigational model definitions use a query language capable of querying both schema and instances, enabling the specification of flexible access structures. Additionally, we propose the use of faceted access structures to improve the selection of navigational objects organized by multiple criteria. Finally, we present an implementation architecture that allows the direct use of the application specifications when deriving a final application implementation.

1. Introduction

According to [1], the Semantic Web is an extension of the current web in which better defines the meaning of the information, enabling computers and people to work better in cooperation. Nowadays, facilities to put machine-understandable data on the web are becoming a high priority for many communities [24]. The web can reach its full potential only if it becomes a place where automated tools as well as people can share and process data.

In this path towards the future Semantic Web, we can notice a lot of different investigations to help make web data available as a large decentralized Knowledge Representation system, using ontology definition languages like DAML+OIL [5], and the “work in progress” W3C Web Ontology Language OWL [20], [23]. We can also observe that research about Web Services is expanding rapidly as the need for application-to-application communication and inter operability grows. As stated in [7], the Semantic Web, like the World Wide Web, can grow from taking well-established ideas, and making them work interoperably over the Internet.

This paper expands and details the ideas presented in [12], adding details to faceted navigation, and presenting new information about implementation architectures.

This work presents an evolution of our web application design method that has been used for several years, showing how it contributes to the definition of applications in the Semantic Web. Our approach is motivated by the fact that we identified the need to describe metadata not only about data, but also about applications that will use the semantically annotated data. In order to describe the metadata application, we developed a method called SHDM (“Semantic Hypermedia Design Method”). The method is based on our experience designing web and hypermedia applications using the Object Oriented Hypermedia Design Method (OOHDM) [16], [17]. During this research, we are interested in using the knowledge represented in ontologies and our main interest is in processing this knowledge, taking advantage of the new infrastructure being built for the Semantic Web.

The SHDM Method is a model-driven approach to design web applications using five steps: Requirements Gathering, Conceptual Design, Navigational Design, Abstract Interface Design and Implementation. Each step focuses on a particular aspect and produces models, describing details about an application to be run on the web. SHDM keeps the separation between conceptual and navigational design, which is an important cornerstone of OOHDM. By explicitly separating conceptual from navigation design, we address different concerns in web applications. Whereas conceptual modeling and design must reflect objects and behaviors in the application domain, navigation design aims at organizing the hyperspace, taking into account users’ profiles and tasks. Navigational design is a key activity in the implementation of web applications, and we advocate that it must be explicitly separated from conceptual modeling [16]. In SHDM, the navigational design step produces expressive models capable of representing web applications, and even families of web applications.

In order to present our approach, the paper is organized as follows: Section 2 describes the main concepts of the SHDM method; Sections 3 and 4 details two important steps: Conceptual and Navigational Design, showing a summarized example of the method using an Art Ontology and Section 5 presents our concluding remarks.
2. SHDM Method Overview

SHDM is a model-driven approach to design web applications using five different steps: Requirements Gathering, Conceptual Design, Navigational Design, Abstract Interface Design and Implementation. Table 1 presents the artifacts produced by each phase. Due to space restrictions, we will not explain every step, but focus on the two most important ones: conceptual and navigational design. Additional details can be found in [11].

SHDM maintains the separation between conceptual and navigational design, originated in OOHDM, enriching the models with several new mechanisms, inspired by the languages being proposed for the Semantic Web.

The information items described in the Conceptual Model and in the Navigation Class Schema are resources manipulated in Semantic Web languages, such as the W3C Resource Description Framework (RDF) [10]. SHDM characterizes resources using ontology definition languages such as DAML+OIL and OWL, expressing advanced aspects such as constraints (restrictions), enumeration and XML Schema datatypes.1

<table>
<thead>
<tr>
<th>Steps</th>
<th>Artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Gathering</td>
<td>Scenarios; User Interaction Diagrams; Design Patterns</td>
</tr>
<tr>
<td>Conceptual Design</td>
<td>SHDM Conceptual Model SHDM composed by:</td>
</tr>
<tr>
<td></td>
<td>• SHDM Conceptual Schema;</td>
</tr>
<tr>
<td></td>
<td>• SHDM Conceptual Ontology;</td>
</tr>
<tr>
<td></td>
<td>• Instances</td>
</tr>
<tr>
<td>Navigational Design</td>
<td>SHDM Navigational Model composed by:</td>
</tr>
<tr>
<td></td>
<td>• SHDM Navigational Class Schema;</td>
</tr>
<tr>
<td></td>
<td>• SHDM Navigational Context Schema;</td>
</tr>
<tr>
<td></td>
<td>• Specification Cards describing:</td>
</tr>
<tr>
<td></td>
<td>• Contexts, Access Structures and Facets;</td>
</tr>
<tr>
<td></td>
<td>• SHDM Navigational Ontology</td>
</tr>
<tr>
<td>Abstract Interface Design</td>
<td>Abstract Data Views; Configuration Diagrams; ADV-Charts; Design Patterns</td>
</tr>
</tbody>
</table>

1 In our examples we used DAML+OIL only because OWL in still at “W3C Working Draft” status, i.e., not completely specified as a W3C Recommendation. Since OWL is partially derived from the DAML+OIL, we plan to make the adaptations as soon as it reaches a more mature status.

| Implementation          | Running application using the previous artifacts and the mechanisms supported by the target environment (parser, inference engine, Java classes, .jsp pages, etc) |

Table 1. SHDM steps and artifacts

We would like to stress that, even though we know that the Semantic Web underlying framework (RDF) is not object-oriented (OO), we still find it useful to use some of the OO modeling principles, mainly because it allows suppressing details by grouping of descriptions, providing a higher level of abstraction.

In order to represent the SHDM navigational model, we need the assistance of a query language capable of dealing with XML. However, simply querying the XML files at a syntactic level or at a structure level would not extract the kind of semantic information we require. Therefore, we defined the following pre-requisites for the semantic query language we need:

- capability of extracting sets, to represent the contexts (see Sec. 4);
- capability of querying both schema and instance, to represent contexts and groups of contexts, among other uses;
- inference capability, to extract information that was not previously modeled;
- declarative mode, to allow the description of the query in SHDM specification cards;
- XML Schema datatype support.

We chose to use the RQL query language [9], a typed language following a functional approach. It supports generalized path expressions featuring variables on both labels for nodes (i.e., classes) and edges (i.e., properties). RQL relies on a formal graph model that captures the RDF modeling primitives and permits the interpretation of superimposed resource descriptions by means of one or more schemas. The novelty of RQL lies in its ability to combine schema and data querying smoothly. RQL supports: XML Schema data types (for filtering literal values), grouping primitives (for constructing nested XML results), arithmetic operations (for converting literal values), aggregate functions (for extracting statistics), namespace facilities (for handling different schemas), meta-schemas querying (for browsing schemas) and recursive traversal of class and property hierarchies. It should be stated that, up to the time of writing of this paper, RQL was the only language capable of querying both schema and data definitions.

In the following sections, we detail some novelties (with respect to established application design methods) of two main SHDM activities, namely, Conceptual Design and Navigational Design. Section 4 outlines the Implementation step.
3. SHDM Conceptual Design

During the SHDM Conceptual Design step we build a model (the Conceptual Model) showing classes and their relationships specifically related to a domain. Classes are described as in object-oriented (OO) UML models [14], with three distinguished details on attributes: they can be multi-typed (representing different perspectives of the same real-world entity), they are described with multiplicity (referring to the number of times the attribute may occur in instances), and they can have explicit enumerations (defining the possible values for that attribute in instances). Relations are described also as in OO UML models, with one additional detail: relations can be specialized creating subrelation hierarchies. The conceptual model obtained using the UML class diagram can be mapped to a RDF/XML serialization format according to heuristic rules [11] summarized next.

When comparing the object-oriented model (OO) with the RDF model it is possible to state that the concepts of classes and subclasses (specialization and generalization relations) can be modeled equivalently. However, there is a significant difference in modeling OO attributes and OO association relations in the RDF model. In RDF models, there is no distinction between a property that describes a class (attribute) and a property that describes an association relation with another class. In addition, RDF properties can be specialized through subsumption relations, allowing the creation of subproperties. Our Conceptual Schema takes advantage of these characteristics as shown below; for reasons of space, only the main ones are detailed.

Every class is mapped to a DAML+OIL Class, modeling attributes and relationships as properties. We use DAML+OIL extensions defined as daml: DatatypeProperty and daml: ObjecttypeProperty to represent attributes and relationships, respectively. Attribute multiplicity is mapped to daml:minCardinality and daml:maxCardinality on specific properties. Attribute enumerations are mapped to the constructor daml:oneOf, providing a means to define a class by direct enumeration of its members, in such a way that no other individuals can be declared as belonging to the class. Datatypes are defined as in XML Schema.

We developed a case study that we called Art ontology, inspired by a Museum example found in [9], which we sketch it here. Our goal was not to be exhaustive in our modeling, but to focus on illustrating the novelties in SHDM schemata.

Figure 1 presents the Conceptual Model for an Art Ontology where the superclass Artist has an association relation ("creates") with the superclass Artifact. There are two specializations for this relation: "paints" and "sculpts," relating the respective subclasses. This means that whenever somebody paints something, he/she is also creating it. This also means that whenever we query the instances of this model to ask for domain and range of the "creates" relation, we will also retrieve the union of the subrelations.

In addition to defining classes and instances declaratively, DAML+OIL and other Description Logics languages let us create intensional class definitions using boolean expressions specifying necessary, or necessary and sufficient conditions for class membership. These languages rely on inference engines (classifiers) to compute a class hierarchy, and to determine class membership of instances based on the properties of classes and instances [13]. SHDM incorporates these Semantic Web languages approaches using Inferred Classes, represented graphically as UML stereotypes (see Fig. 2). One advantage of using the stereotype notation in the conceptual model is the ability to distinguish these classes from the ones that are explicitly declared as subclasses.

Fig. 1. Art Conceptual Class Schema

Fig. 2. Inferred Class, with two alternative DAML+OIL equivalent definitions

This simple example states that a “Awarded” is a subclass of Painter if the property value of his/her awarded attribute is “true” or that the “Painter” subclass “Awarded” is the intersection of classes “Painter” and the set of resources whose “awarded” property satisfies the condition of having its value equal to “true”. Since this language relies on inference engines to compute a class hierarchy, we can validate our model using any DAML+OIL inference engine. Other than the inferred
classes, we defined in our metamodel another stereotype, to represent class hierarchies with arbitrary depth, called “arbitraryClassHierarchy,” but due to space restrictions, we will not detail it in this article. This stereotype corresponds to the RQL query that retrieves a complete class hierarchy, given a root class.

4. SHDM Navigational Design

An important tenet of OOHDM, followed by SHDM, is the realization that navigation objects are actually views over conceptual objects [16]. The SHDM Navigational Design defines a navigational vision of the Conceptual Design, specifying the information that will be processed, and the possible navigations among them, according to user profiles and tasks to be supported. During this step, we define the information items that will be manipulated by the user, with the following subtlety: we define “What will be manipulated” and not “How it will be manipulated.”

- During the navigational design we are interested in specifying:
  - which objects can be reached by the user (the navigational nodes);
  - which relations exist among these navigational nodes (the links);
  - within which sets of objects the user will navigate (the contexts);
  - in which ways these sets will be accessed (the access structures);
  - which different content must be presented to the user, depending on the context he is in (the inContext classes).

The main SHDM navigational primitives are navigational objects (nodes), navigational contexts and access structures. Navigational objects are defined as views over conceptual objects, and can be either navigational nodes, links or inContexts classes. Navigational contexts are sets of navigational objects that follow rules specified by the application designer (ex: order). The access structures are also sets of navigational objects with a specific order, but with a peculiarity, each object in these sets have at least one reactive selector that activates a link to another object.

The Navigation Design activity generates two schemas: the Navigational Class Schema and the Navigational Context Schema. The first defines all navigable objects as views over the application domain. The navigable relations are links between nodes and also the new subrelations that allow a new type of navigation based on subsumption relations between links. The second schema defines navigational contexts (the main structuring primitive for the navigational space), access structures used to reach these contexts and links that connect them.

4.1 SHDM Navigational Class Schema

After completing a SHDM Conceptual Model with the classes and relations that were considered relevant to a domain, we model the navigation with the information items that can be navigated by the user. In Fig. 3 we use a Navigational Class Schema to describe the Art ontology example.

While seeing the node 2 Artifact the user can perceive attributes that did not belong to the original Conceptual Class, such as the style of the Artifact. An anchor is available to allow the navigation to that specific Style node, in case the user wants to know more about that Style.

Each instance of Artist is presented with an index of his/hers Artifacts. The user will be able to choose any Artifact in this index and move to another context: the context of all the Artifacts that specific Artist made. Inside this context s/he will navigate from previous to next objects according to the order defined by the designer (among several like “by creationDate ascending”, “alphabetic ascending,” etc.). In an equivalent way, each time an instance of Artifact is presented, the user will be able to see an anchor that can be selected to take him to a context of Artists in order to see more information about the specific Artist that created the chosen Artifact.

---

The node is the Navigational Class and not the Conceptual Class.
classes. The mappings are specified using an RQL [3] query, as exemplified below.

**RQL mapping:**

<table>
<thead>
<tr>
<th>RQL query</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>select y from {Artist} firstName {y}</code></td>
<td>retrieves the firstName of Artists</td>
</tr>
<tr>
<td><code>select y from {x} creates {y}</code></td>
<td>retrieves all Artifacts of a specific Artist</td>
</tr>
<tr>
<td><code>where x= &quot;parameterA&quot;</code></td>
<td></td>
</tr>
</tbody>
</table>

As in the conceptual schema, SHDM also allows subrelations in the navigational class schema. In addition to using subrelations defined in the conceptual model, it is possible to use subrelations in the mapping of the conceptual model into the navigational class model. For example, it is possible to define navigational subrelations of “creates” by restricting its subclasses, for instance, only those whose counter-domain is a specific subclass of Painting, such as Watercolor. In Figure. 4 we illustrate a combination of DAML+OIL and RQL to specify the mapping from the conceptual to the navigational ontology (identified with namespace shdm).

```xml
<daml:ObjectProperty rdf:ID="etches">  
  <daml:subPropertyOf rdf:resource="#creates"/>
  <shdm:rqlQuery="select y from {x} creates (y); where x= "cult:Watercolor"/>
  <daml:range rdf:resource="#Painter"/>
  <daml:domain rdf:resource="#Watercolor"/>
</daml:ObjectProperty>
```

**Figure. 4. Navigational Class attribute mapping using subrelations**

The Navigational Class Schema is useful when designing a web application, because it allows the designer to define all navigable classes and the links between them. However, our experience proved that another schema is also necessary, and complementary: the Navigational Context Schema, described next.

### 4.2 SHDM Navigational Context Schema

In the SHDM Navigational Context Schema, the designer can detail two important aspects:

- the different ways how objects can be grouped during navigation, and
- the access structures to reach these objects.

Navigational Contexts remain a very important navigational primitive in our approach, since it allows us to describe sets of navigational objects relevant to the user during a task. The novelty lies in the fact that the language used to define contexts is more expressive than the previous one.

A context groups objects related to each other by some aspect (e.g., common attributes or being related to a common object), and organizes these objects as sets of nodes, defining in which way they may be accessed (e.g., sequentially).

Navigation contexts may be further specified as groups of contexts, since sometimes it is possible to parameterize their defining property. For example, “Sculpture By Material” is actually a set of sets; each set is a context, determined by one value of the “material” attribute. There is an analogous definition for contexts whose property is based on 1-to-n relations, such as “Sculpture By Sculptor.”

Contexts play an analogous abstraction role in defining the navigational behavior as classes play in defining objects. All items in a context exhibit similar navigation properties, and therefore navigation can be entirely characterized by specifying navigation contexts, as opposed to individual navigation items.

Access structures are indexes (collections of links) that allow the user to reach navigation objects (within some context). SHDM allows defining both Access Structures and Navigational Contexts using meta-data properties. The `<<subClassOf>>` stereotype indicates that the corresponding element (access structure or navigational context) is a set of elements, one for each sub-class.

**Figure. 5. Access Structures and Navigational Contexts defined based on meta-model properties**

In Figure. 5 we show the graphical notation and the RQL statements for an example. In the artists access structure we represent a list of links to all artists (the order is defined in the corresponding specification card). The inner dashed box represents subsets of artists defined according to subclasses, for example Painter or Sculptor.

The context “Artifact By Artist” is composed of all artifacts created by a specific artist. This context can be accessed by choosing an artist as a parameter of selection. The innermost box signifies that the user can also choose any subclass of Artist to group the artifacts.

A very common situation found in application domains is when the user must start a task by some kind of search, which can be performed according to various possible criteria. Furthermore, there is no a-priori order of criteria that is deemed best. For example, consider the task of choosing an artifact in a museum – there are multiple criteria that can be employed, such as “style,” “location,” “period,” etc… Each user may choose a different order (strategy) to narrow the search. Notice that in many situations, a direct query is not acceptable, as the
user may want to browse through intermediate results of the filtering process.

The use of facets will handle this type of situation. Facets were initially proposed in library and information sciences [15]. Simply put, a facet can be considered as a category. In [21] Taylor defines facets as “clearly defined, mutually exclusive, and collectively exhaustive aspects, properties, or characteristics of a class or specific subject.”

In a faceted classification scheme, the facets may be considered to be dimensions in a cartesian classification space, and the value of a facet is the position of the artifact in that dimension. Within each facet, subfacets or topics that are more specific are listed. The breakdown continues into subfacets within subfacets.

For SHDM, we take advantage of the increased availability in the WWW of taxonomies that can be used as facets. For this, we created a new primitive, named Faceted Access Structures, based on the facet concept. We define facet hierarchies based on our navigational attribute types – which are in fact metadata about our Web application. Each hierarchy is defined independently, in order to organize content along a particular dimension.

Faceted Access Structures and Faceted Navigational Contexts are defined using the <<facet>> and <<ByValidFacetComb>> stereotypes. In Figure 6 the outer dashed box denotes the valid combinations of facets to reach the Artifact navigational class, and the three inner dashed boxes indicate the possibility of choosing just one of the facets. The context “Artifact By ValidFacetCombination” stands for the possibility of accessing Artifacts by any combination of Region or Style. Similarly, “Artifact By Style <<facet>>” stands for all sets of artifacts grouped by Style and by its subclasses.

Figure 6. Faceted Access Structures and Faceted Navigational Contexts

Figure 7 details the specification cards corresponding to the faceted elements. The designer can use a graphical notation to annotate in the facet hierarchies numbers (as labels) that represent the invalid combinations. When the designer describes the combinations, he/she does not have to make it extensively; it is enough to only annotate the nodes that are superclasses of the invalid combinations, at any level of the hierarchies. An algorithm such as proposed by Tzitzikas in [22] can generate the enumerated combinations.

Figure 7. Navigational Faceted Access Structures Specification Card

Figure 8 shows the specification corresponding to this facet structure in RDF.

To specify the invalid combinations, the SHDM designer should only specify the root of each Facet that may generate an empty set. This root can be either a FacetComposite or a FacetLeaf.
leveraging classification initiatives already made for particular knowledge domains, such as Geographical Information, Medicine, e-Commerce, etc…

In Figure. 9 we exemplify some of the new primitives in the Navigational Context Schema. Web application modeling using OOHDM explicitly represents each context reachable by the user, improving understanding between the designer and the user. With SHDM, we adopted additional conceptual semantic modeling primitives, and a richer query language such as RQL, making it possible to specify concisely a larger and more complex set of contexts with simple expressions.

To improve understandability of context diagrams, we adopt the notation <<subclassOf>> to represent the set of contexts determined by the members of the indicated subclass. We also provide a similar abstraction for the access structures, representing groups of access structures organized by subclasses.

![Figure. 9. Art Navigational Context Schema](image)

The abstraction power of the notation proposed is exemplified in several places. The use of compact facet specifications avoids explicit enumeration of all possible combinations, including those not known at design time. The same is true for the use of the <<subclassOf>> stereotype, as it allows definitions of access structures and contexts for an arbitrary class hierarchy. Having used RQL, we are able to query both data and metadata. For instance, we can now define a context “Artifact By Style”, without knowing ahead of time all possible values (or subclasses) of “Style”. If the designer later adds a new subclass to “Style,” and its corresponding instances, the same application specification still applies. In this sense, SHDM specifications could be regarded as specifying application frameworks (as in [18]). Although not shown here, similar reasoning can be applied to inferred classes.

In order to clarify the abstraction power of the notation proposed, we use the art example to present a comparison of two navigational context schemas: one modeled using SHDM (Figure. 9) and the equivalent one using OOHDM (Figure. 10). To help describe the figures we divided them in three parts. The top part (numbered as one) shows the use of faceted index navigation.

![Figure. 10. OOHDM Art Navigational Context Schema](image)

In Figure. 10 following a step-by-step navigation in the part of the diagram labeled “n”, we can simulate a user that wants to start a navigation by first choosing, among several regions, one such as South America. Then he could still want to drill down in the region hierarchy (to what we could call here sub-regions) choosing a country such as Brazil, then a state as Rio de Janeiro, and so on. At any point in this navigation, he could have decided to stop navigating through regions and go directly to the context of “Artifacts By Region” (using as parameter the last chosen region). There he would see all Artifacts created in that specific region. In Figure. 10, we can see that this would require many paths whereas in the concise version (Figure. 9) we were able to represent all options of navigation through faceted access structures with one line (faceted access structure “Styles” taking the user to context “Artifact By Style”).

Still in example number one, the user could have preferred to switch from navigating through Regions to a
navigation through Styles at any point. We can observe how hard it is to express all the possibilities in the expanded version.

Now examining the middle part of the diagram (labeled “2”), we can compare the navigation through access structures and their subclasses. For example, let us imagine a user that wants to see Artifacts By the museums where that exhibited them. He can start navigation by first looking at an access structure (for instance a selectable list) of all the museums. Then after choosing one museum he can see another access structure of all artifacts exhibited there, ordered alphabetically. However, he might prefer to see only the paintings and not the sculptures. He could then navigate in the subclass hierarchy and choose any subclass of artifact, and from there he could reach the context of “<<subclassOf:Artifact>> By Museum.”

The bottom part (labeled “3”), can be described as the navigation through the contexts of Artifacts By Artist, where the user can see all artifacts created by an artist, all paintings painted by a painter, all sculptures sculpted by a specific sculptor, and so on. The simplest way to reach the “Artifacts By Artist” context is by just selecting one among a list of artists. However, the user could be interested in seeing a list of painters instead, to help him narrow his search. Even then, he could want to see all paintings made by cubists grouped by their painters. This is exactly what we are expressing in Figure. 10, in part labeled “3,” by using the parallel paths where the first reaches the access structure Artists and then the context “Artifact By Artist,” and the second shows the navigation through access structures that can be any subclass of Artist chosen, and then leading the user to the context of “Artifacts By” that same subclass.

The last group of navigations specified is the one where a user wants to see information about artists. He might want to choose an artist and navigate straight to a context of artists ordered alphabetically, or he might prefer to see all them grouped as painters, cubists, etc. The two bottommost arrows in Figure. 9 express this navigation, being equivalent to the five bottommost arrows in Figure. 10.

4.3 Current Implementation

We are developing an implementation environment depicted in Figure. 11. We are now focusing on the storage, inference and query environment, using the Sesame architecture [4] to store SHDM conceptual and navigational ontologies.

This environment, developed in Java, offers RDF and RDF(S) [3] storage and RQL support. More specifically, we are now testing a DAML+OIL reasoner called BOR [19] integrated with Sesame through an additional inference layer, capable of dealing with DAML+OIL ontologies. The inferences occur at schema and instance levels, thus, new tuples are obtained and persisted in the repository. The RQL query language can then still take advantage of querying both schema and instances, since the repository stores RDF and RDF(S). Figure. 12 shows an extension of the BOR architecture that is currently being used.

![Figure 11. Implementation architecture for SHDM designs.](image)

We already implemented the Art ontology presented in this article and deployed an initial version where the ontologies produced during the conceptual and navigational designs were used in the implementation phase.

![Figure 12. Implementation for the Storage, Inference and Query Component.](image)

The implementation uses page templates, as shown in Figure. 14. Although not discussed in this paper, the
concrete interface specification is derived from an abstract interface specification, where the major interface elements are specified in terms of their interaction capabilities. The concrete implementation is responsible for the ‘look and feel’. Therefore, the template includes abstract elements such as \(<\text{Active Reference}>\), which will generate a logical path to the current page, with references to relevant intermediate points, for example, Artists—Painters—Cubists. Similarly, \(<\text{prev}>\) and \(<\text{next}>\) generate links to the previous and next elements within the current context. The \(<\text{chooser}>\) element generates an interface widget that allows the choice between the various elements.

In this paper, we have argued that web application design methods can benefit from modeling language primitives being proposed for the Semantic Web, such as RDF, RDFS, and DAML+OIL. Some approaches, such as HERA [6] propose directly using RDF and RDFS, or slight extensions, as the basic ontology modeling language, equivalent to our conceptual modeling. Others, such as OntoWebber [8], add additional ontologies on top of them, to cover other aspects of application design, such as site structure. In contrast, we have kept the traditional UML-like object model, extending it with a few primitives such as subrelations, taken from RDF, and anonymous classes defined through restrictions, taken from DAML+OIL.

We have followed the original OOHDM approach of defining the Navigational Class model as a mapping of the Conceptual Model, but using RQL as the mapping specification language, which is able to query DAML+OIL models. Another benefit brought by SHDM is the ability to specify concisely faceted navigation structures. It was shown how facet specification is equivalent to very large enumerations of possible navigations paths. With the increasing availability of domain taxonomies, this will allow easily incorporating such taxonomies as part of the navigation structure of applications designed using SHDM. In addition, the resulting applications are able cater to varying user profiles by providing alternative navigation paths better suited to each particular case.

An interesting issue that arises relates to what will the applications in the Semantic Web be like. As more and more metadata is associated to resources, much of the current linking will become links defined or inferred in (or through) the metadata. In fact, the metadata should encode most of the semantically important information. Therefore, it is reasonable to suppose that future applications will exhibit a balance between information conveyed in resources, and information conveyed in the metadata.

For current applications, there is some redundancy between contents of documents and metadata; for instance, the name of a painter is likely to appear on a web page about a painting, and it might even be a link. This same information is probably present in the metadata. In the extreme case, all information is extracted from the metadata, and the application becomes like a database-backed application, except that all the data is stored in RDF triples. Thus, the question remains, what is the proper balance? This is one of our current directions of investigation.

Another area of investigation is the integration of existing taxonomies with the conceptual and navigation designs. This area will benefit from the work already in progress regarding integration of ontologies.

5. Conclusions
We are also investigating how to extend SHDM for personalized and adaptable web applications. We are also pursuing additional topics such as integration of interface and interaction models, and of application functionalities.

6. References


