Specification Framework for Engineering Adaptive Web Applications

Flavius Frasincar, Geert-Jan Houben, Richard Vdovjak
Eindhoven University of Technology
Department of Mathematics and Computer Science
PO Box 513
5600 MB Eindhoven, The Netherlands
{flaviusf, houben, richardv}@win.tue.nl

Abstract

The growing demand for data-driven Web applications has led to the need for a structured and controlled approach to the engineering of such applications. Both designers and developers need a framework that in all stages of the engineering process allows them to specify the relevant aspects of the application. This paper concentrates on Web applications that automatically generate hypermedia presentations for their output. Typically, these applications retrieve their data from a heterogeneous set of Web data sources, and they respond to a user's request for information by providing the user with a hypermedia presentation for the requested data. Many classes of Web-based information systems are of this nature. Because of this aspect of automated presentation generation, (the support of) the engineering process for these applications is far from trivial. The engineering becomes even more complicated when we include notions of adaptivity. Here, we address both adaptation during the presentation generation for the sake of personalization (to reflect e.g. user preferences or platform used), as well as adaptation within the generated presentation (generating adaptive hypermedia).

The specification framework that we present in this paper can be used in an engineering process of an adaptive Web application. This framework called Hera is related to design methodologies for Web applications, and in particular to RMM (Relationship Management Methodology) for its coverage of aspects of hypermedia design. Characteristically, Hera distinguishes between the logical or functional specification of the data and its actual presentation. By separating (1) the conceptual (or semantical) description of data, (2) the navigational aspects of its hypermedia presentation and (3) the rendering of the presentation, the process of the design of the application improves significantly. Using new emerging Web technologies like RDF, XML, and XSLT, we have implemented a prototype to illustrate the use of this specification framework.

Keywords: Web Engineering, WIS, Personalization, Adaptive Hypermedia, RDF(S)
Word count: 7980

1 Introduction

Since its birth in the early nineties the World Wide Web has grown into one of the most popular channels for reaching a very diverse audience on different platforms worldwide and 24 hours per day. Its success is overwhelming and its impact on the humankind is tremendous. Some compare its importance with Guttenberg's invention of the printing press. However, the World Wide Web

is becoming a victim of its own success. Ad hoc hacking without a prior design using no rigorous methodology is the common Web development practice of today. This approach fails to meet the growing demand for high quality data driven Web applications such as Web-based Information Systems (WIS) [22].

WIS applications use Web technologies to fulfill the needs of professional information systems. While the engineering and development process for the more traditional, hand-crafted applications on the Web is already stretched to (or beyond) its limits, the specific role of a modern WIS asks for a more highly structured and controlled approach to Web engineering [30]. Many WIS have a data-driven nature, that requires a process of automatically generating hypermedia or multimedia (Web) presentations for the data to be output. This automated hypermedia generation process implies that part of the design decisions need to be specified beforehand, such that later, at run-time, the actual design gets generated.

To facilitate the Web engineering of these data-driven applications there is an obvious need for a design framework. This framework should allow designers to specify and reason about WIS in an appropriate level of abstraction depending on the different stages of the engineering project (requirements analysis, design, and implementation), but also on the different dimensions of the problem area (e.g. data modeling, hyperspace navigation modeling, user/platform adaptation, layout design etc.).

In the context of engineering WIS, we explicitly mention the aspect of adaptation [7]. It is natural to associate the automated process of presentation generation with aspects of adaptation. For example, the user's background and history, and the platform used to view the data could influence the generation process. By acknowledging the user's context the quality (functionality) of the generated presentation can be highly improved. Note that adaptation can play a role in the generation process itself, as well as in the presentation that is generated: both the process and its outcome (the hypermedia presentation) can be adaptive.

The nature of our target applications is such that during presentation generation a number of design decisions are executed in an automated fashion. The specification of these parts of the design process can benefit a lot from technologies introduced by the Semantic Web initiative [4]. Additionally, to blend in WIS applications into the WWW, they should make use of these Semantic Web technologies. This allows that the data that resides on the WWW is not targeted only for humans but it is also machine understandable, thus enabling creation of new services such as smart search engines, presentation generation, automated information integration and exchange etc.

This paper addresses the above issues by suggesting a design methodology as a basis for the Web engineering project, as well as proposing a specification framework (called Hera) supporting this design methodology (including its partial automation). The Hera methodology is inspired by RMM [26] and includes several features in order to provide a better support for an automated design of adaptive Web applications which can be deployed in the context of the Semantic Web. Characteristically, Hera distinguishes between the logical or functional specification of the data and its actual presentation. It separates (1) the conceptual (or semantical) description of data (expressed by the Conceptual Model (CM)), (2) the navigational aspects of its hypermedia

presentation (expressed by the Application Model (AM)) and (3) the (visual) rendering of the presentation. Orthogonally, it also expresses the adaptation aspects in the User Adaptation Model (UAM). These models together form the foundation of our specification framework and this paper focuses on the description of these models (specially CM, AM and UAM), their interrelationships and how their use can be implemented (e.g. authoring support).

The rest of the paper is structured as follows. In Section 2 we provide an overview of related work. In Section 3 we present the Hera design methodology and discuss the individual steps that are involved in this approach[1]. In Section 4 we consider the Conceptual Model, while the Application Model is the subject of Section 5. Section 6 addresses the different aspects of adaptation. After having discussed these models, Section 7 shows an HTML and WML rendering approach based on XSLT. Section 8 concludes with a short summary and a discussion of future work.

2 Related Work

For a long time Web application engineering has been synonymous with ad hoc development and the lack of use of a rigorous design methodology. Now that Web application engineering becomes more mature (or at least, is required to be more mature because of its professional deployment), proposals for design methodologies are advocated. Not just for the sake of a better (and better controlled) application design process, but also to have a means to keep a grip on the different, diverse aspects that play a role in modern Web-based Information Systems. Aspects like data integration and adaptation complicate the design process and bring its complexity beyond the level that is easily handled by a single human developer. Therefore, a strong methodology (supported by a suite of tools) can help to keep the design process at a practical level.

Methodologies like Relationship Management Methodology (RMM) [3, 23, 24, 25, 26] and Object Oriented Hypermedia Design Methodology (OOHDM) [33, 34, 35] have been originally developed to support a hypermedia design process. For that purpose they succeed in identifying the different steps to be taken in the development of Web applications. As is nicely shown in [3] the design of the specific hypermedia aspect of splitting up information into pages and defining the navigation between those pages, requires new ways of specifying this hypermedia metaphor. However, while these methodologies represent an important step forward, they are not particularly applicable in the context of automated hypermedia presentation design, which is demanded in WIS-like applications. The issue of supporting the automated hypermedia design closely resembles the goals presented in [38], where they use the term third generation for this new kind of Web applications developed in the context of the Semantic Web.

Another drawback of methodologies like RMM is that they are not specially targeted at the support of personalization or adaptation.

Personalization means that the application acknowledges the user's situation (e.g. specific user preferences or the platform or device used) and its information delivery is adapted accordingly.

[1]This paper does not go into details of the data integration aspect, which is part of the Hera methodology. Data integration concerns the construction of one set of data available for querying out of the distributed and differently formatted sources [39].

[32] extends OOHDM in order to give personalization the appropriate place in the application development life cycle. [17] shows how also a specific software architecture is needed to achieve personalized content delivery, thus giving a nice example of the use of Web technology to realize personalization in information presentation.

On the other hand, most of the currently existing methodologies lack a user model that would allow for the design of truly adaptive Web applications, where adaptation means that the application changes its content (delivery) based on the state of a user's knowledge or browsing behavior within the application. Adaptation [7] can play a significant role in modern applications, and therefore our aim is to include this aspect not just in the light of the personalization mentioned earlier but also in the context of the hypermedia presentations that get generated[2]. Generating adaptive presentations requires a clean separation of concerns, as is advocated in [8]. That model gives a good theoretical framework, but lacks the necessary specification formalisms needed to achieve concrete designs.

A language like WebML [11] gives a good basis for specifying important aspects of a Web site design. While less ambitious as a complete design methodology, it offers a significant contribution by providing a specification language for the orthogonal dimensions of content structure, page composition, (link) navigation, presentation and personalization.

Another aspect to the issue of engineering support is the fact that the realization of all this functionality usually involves a multitude of different tools. Therefore, the aspect of interoperability becomes an issue, which in turn implies the desire to use a suitable exchange format for the data involved. In [28] XWMF is proposed as a way to use RDF [29] as an exchange format dealing with the metadata that is essential for strong interoperability.

3 Hera Specification Framework

The primary focus of the Hera project [20] is to provide engineering support for data-driven Web applications that automatically generate hypermedia presentations in response to ad hoc user queries.

The Hera design methodology guides the designer through the different steps of the design process, each of them yielding a specification (model) that is being interpreted (executed) by the Hera suite to achieve the objective of automatic presentation generation.

3.1 Hera Design Methodology

The Hera design methodology has its origins in RMM. Similarly to RMM it distinguishes several steps to be followed during the design process of a Web application. Each design step produces as its outcome a specification with a certain level of abstraction based on the separation-of-concerns principle. The sequence of these steps is depicted in Figure 1[3].

[2]Note that in the field of adaptation sometimes a difference is made between these two kinds of adaptation. Generating a different application based on the situation implies that the application is "adapted", while making the application itself change its content delivery based on the user's browsing behavior within the application implies that the application is "adaptive".

[3]Note that the arrows in the picture denote only the general flow: it is possible to have feedback loops in the process.

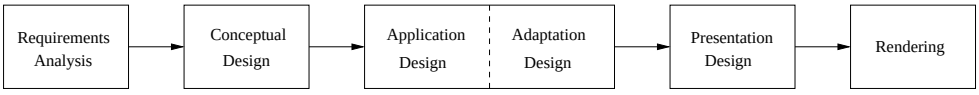


Figure 1: Design Methodology

Although requirements analysis and presentation design are important phases in the Web engineering life cycle, they are beyond the scope of this paper where we primarily concentrate on the steps describing conceptual, application, and adaptation design.

In the Conceptual Design step the application domain is represented using traditional conceptual modeling techniques. At this phase a Conceptual Model (CM) consisting of a hierarchy of concepts, their properties, and relations is constructed. Within the information system one could distinguish a data repository that contains the data that is available for use in the presentations to be generated: the queries are asked against this repository. The purpose of the CM is to describe which information is available inside this data repository. Note that the aspect of data integration (outside this paper's scope) strongly influences this model.

In the Application Design step, the concepts from the CM are transformed into slices. In analogy to RMM we use the notions of slice and slice relationship to model how the concepts from the CM will be presented. While the CM gives a *semantic description* of the information that is available, the Application Model (AM) gives a *navigational description* of that information. Having in mind that hypermedia presentations will be generated for queries against this information, the navigational description specifies (a blueprint for) the hypermedia nature of those presentations. So, the AM does not yet include all the rendering details covered in the next step of the methodology, but it does allow for a first, logical sketch of the hypermedia nature of the presentation to be generated. This implies the translation of concepts into "pages" and of establishing a navigational structure between those pages.

Whenever adaptation is an issue for the application, its place in the specification design is in this phase: Adaptation Design is associated with Application Design. As a consequence, a User Adaptation Model supporting the adaptation is built during Adaptation Design. In line with the reference model from [8] this UAM includes the specification of a user model and adaptation rules.

The Presentation Design step introduces an implementation independent Presentation Model (PM) [4] , which focuses on layout, hyperlinking, timing and synchronization issues. So, Presentation Design elaborates on the Application Design and bridges that to the subsequent phase of Rendering (which does the actual code generation).

[4]Description of the Presentation Model is outside the scope of this paper, interested readers are referred to [19].

3.2 Hera Suite

The Hera suite is a collection of engines (software programs), which interpret the specifications (or models) provided by the designer during the different phases of the design process. The suite is split into several layers that each process a different model and thus reflect a different design phase. The different layers of the suite are depicted in Figure 2.

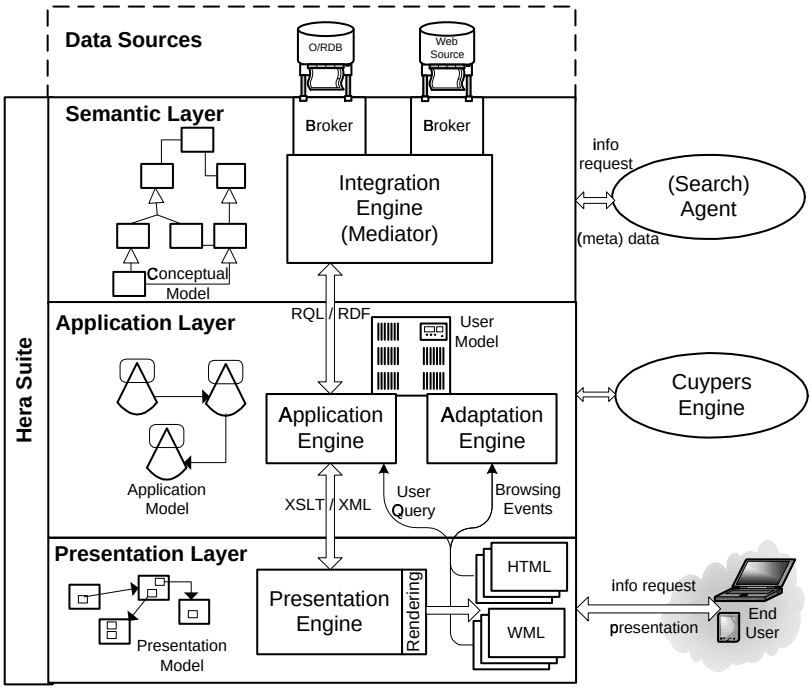


Figure 2: Hera Suite

- The Semantic Layer integrates the data that is available in a collection of heterogeneous external data sources into one Conceptual Model (CM) with well-understood semantics. That model specifies the data available (from the sources) for the Web application: it describes a (virtual) data repository. Since the on-demand retrieval paradigm was chosen to realize the integration, the external sources are consulted each time there is a data request. Therefore, there is no need for a centrally materialized data repository in our suite [39]. By constructing this repository the Conceptual Model introduced by the designer in this layer represents a semantically unified interface for querying (selections of) the heterogeneous sources. Note that we do not aim at merging all possible sources together in order to provide a cumulated view of all attributes of the data. We argue that such an approach offers weak semantics, where the understanding of the semantic structure of the integrated sources is effectively left up to the user who is asking the query.
- The Application Layer provides the logical functionality of the application and bridges the

Semantic Layer and the Presentation Layer. In the context of our target applications it is not feasible to manually design a hypermedia presentation for each user query. Instead, a global Application Model is designed and each query is expressed by the user against this model. The Application Model specifies the hypermedia aspects of the application. It describes how the data is presented and accessed: stated differently, it specifies how the user can navigate through the data. The data is grouped together into meaningful presentation units, called slices: these slices act as the units of navigation ("pages"). Slice relationships describe relations between slices providing a means to navigate between slices and thus to access information belonging to different presentation units. The Application Layer builds on the Semantic Layer, in the sense that it exploits the structure and attributes of concepts (from the CM) and the relationships among them, and it lays a foundation for the hypermedia nature of the output.

User adaptation is another important issue, which is considered in this layer. The adaptive behavior of the application is captured by the designer in the User Adaptation Model a part of which is embedded in the Application Model (as conditional slices). This part of the UAM is interpreted by the Application Engine and the remaining part (e.g. rules that update the user's state) is interpreted by the Adaptation Engine (during the browsing process).

- The Presentation Layer of the framework is responsible for the transformation of the Application Model (possible including the User Adaptation Model) into a chosen implementation platform (e.g. HTML [2], WML [18], SMIL [14], etc). In order to achieve this an implementation independent presentation specification provided by the designer in the form of the Presentation Model is used.

The W3C standard RDF(S) [10, 29] was chosen to articulate all the models, which are being used. This provides interoperability in all layers of our suite, which is needed in order to open our framework for "the outside world". As shown in Figure 2 there can be a direct information request from a search agent. These kinds of data requests are fulfilled directly by the Semantic Layer. Another example for interoperability is to connect the Cuypers presentation generation engine [38], which builds its own sophisticated presentation, to our Application Layer.

4 Conceptual Model

The Conceptual Model (CM) provides a uniform interface to access the data integrated within a given Web application. It corresponds to a mediated schema that is filled with data during query resolution. The application user is assumed to be familiar with the semantics of terms within his field of interest, and the function of the CM is to offer the user a uniform semantic view over the different sources, that usually use different terms and/or different semantics.

The CM consists of hierarchies of concepts relevant within the given domain, their properties, and relations. As already mentioned above it is expressed (as well as the other models in our framework) in RDF(S) [10, 29]. There are several reasons why we chose to use RDF(S):

- Compared to a traditional database schema, it deals better with the semi-structured nature of Web data.
- Since RDF(S) is a W3C standard for metadata, it brings us closer to semantic interoperability.
- On top of RDF(S) high level ontology languages (e.g. DAML+OIL [37], OIL [16]) are (becoming) available, which allow for expressing axioms and rules about the described classes giving the designer a tool with larger expressive power. Choosing RDF(S) as the foundation for describing the CM enables a smooth transition in this direction.

Although RDFS seems to be a promising modeling tool it does not provide all modeling primitives we demand. Namely the notion of cardinality and inverse relationship is missing and there is also a lack of basic types [21]. On the other hand RDF(S) is meant to be extensible[5], so we can provide our extensions. Figure 3 introduces the notion of *cardinality*, which is a property of relationship (property in RDF) and can be either single or multiple. The *inverse* relationship is defined similarly as a property of a property[6].

```
<rdfs:Class rdf:ID="Cardinality"/>
<Cardinality rdf:ID="single"/>
<Cardinality rdf:ID="multiple"/>
<rdf:Property rdf:ID="cardinality">
    <rdfs:domain rdf:resource="#Property"/>
    <rdfs:range rdf:resource="#Cardinality"/>
</rdf:Property>
<rdf:Property rdf:ID="inverse">
    <rdfs:domain rdf:resource="#Property"/>
    <rdfs:range rdf:resource="#Property"/>
</rdf:Property>
```

Figure 3: System Extensions to RDFS

Figure 4 shows an excerpt of the type extensions we use to model the CM. Here we introduce some basic types like *String* and *Integer*, but also some multimedia types like *Image*, properties (e.g. height and width of an image, duration of a video etc.) of which can prove helpful later during the Presentation Design (rendering) stage. All these types are subclasses of the *Media* type, which has a property called *data* that specifies the actual textual (*String*) content of a concrete instance[7].

In [21] it is shown how to combine the data semantics, expressed in RDF(S), with the data constraints, modeled in XML Schema [5, 15, 36]. As there is not yet a clear W3C Recommendation on this subject we chose the purist approach to model both data constraints and data semantics in RDF(S).

[5]We expect that in time there will be some standard extensions in a library-like format.

[6]We acknowledge that DAML+OIL provides support for this, but at the time of development of the prototype this functionality was not available.

[7]In case of the *Image* type this string is displayed, if the image itself cannot be retrieved.

```
<rdfs:Class rdf:ID="Media"/>
<rdfs:Class rdf:ID="String">
    <rdfs:subClassOf rdf:resource="#Literal"/>
    <rdfs:subClassOf rdf:resource="#Media"/>
</rdfs:Class>
<rdfs:Class rdf:ID="Integer">
    <rdfs:subClassOf resource="#Literal"/>
    <rdfs:subClassOf resource="#Media"/>
</rdfs:Class>
<rdfs:Class rdf:ID="Image">
    <rdfs:subClassOf resource="#Media"/>
</rdfs:Class>
...
<rdf:Property rdf:ID="data">
    <rdfs:domain rdf:resource="#Media"/>
    <rdfs:range rdf:resource="#String"/>
</rdf:Property>
...
```

Figure 4: Media Type Extensions to RDFS

The running example that we use through out the paper describes the design steps of an adaptive Web museum application. The Conceptual Model of our application roughly corresponds to the actual museum catalogue of the Rijksmuseum in Amsterdam.

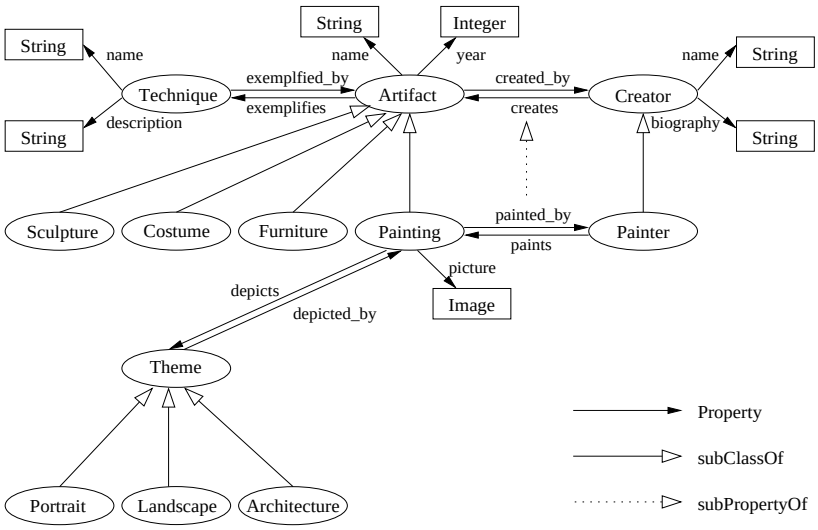


Figure 5: Conceptual Model

A part of this CM is depicted in a graphical notation in Figure 5:

- Concepts are depicted as ovals; the more abstract ones such as *Artifact* and *Creator* are on the top part of the model. From those, more concrete concepts are inherited, e.g. *Painting* and *Painter*.

- Relationships are denoted with full arrows and are modeled as RDF properties. Similarly to concepts there are abstract relationships (properties) such as *created_by*, *creates* and their more concrete subrelationships (subproperties) *painted_by* and *paints*.
- Attributes, e.g. *Creator.name: String*, are treated as properties having as its domain a concept, e.g. *Creator* and as its range a basic type defined in our type extension in Figure 4, e.g. *String* (indicated with the rectangle)

```
<rdfs:Class rdf:ID="Artifact"/>
<rdfs:Class rdf:ID="Painting">
    <rdfs:subClassOf rdf:resource="#Artifact"/>
</rdfs:Class>
<rdf:Property rdf:ID="name">
    <rdfs:domain rdf:resource="#Artifact"/>
    <rdfs:range rdf:resource="#String"/>
</rdf:Property>
<rdf:Property rdf:ID="year">
    <rdfs:domain rdf:resource="#Artifact"/>
    <rdfs:range rdf:resource="#Integer"/>
</rdf:Property>
<rdf:Property rdf:ID="picture">
    <rdfs:domain rdf:resource="#Painting"/>
    <rdfs:range rdf:resource="#Image"/>
</rdf:Property>
<rdf:Property rdf:ID="created_by">
    <sys:cardinality rdf:resource="#single"/>
    <sys:inverse rdf:resource="#creates"/>
    <rdfs:domain rdf:resource="#Artifact"/>
    <rdfs:range rdf:resource="#Creator"/>
</rdf:Property>
<rdf:Property rdf:ID="exemplifies">
   <sys:cardinality rdf:resource="#single"/>
   <sys:inverse rdf:resource="#exemplified_by"/>
   <rdfs:domain rdf:resource="#Artifact"/>
   <rdfs:range rdf:resource="#Technique"/>
</rdf:Property>
<rdf:Property rdf:ID="painted_by">
    <sys:cardinality rdf:resource="#single"/>
    <sys:inverse rdf:resource="#paints"/>
    <rdfs:domain rdf:resource="#Painting"/>
    <rdfs:range rdf:resource="#Painter"/>
    <rdfs:subPropertyOf rdf:resource="#created_by"/>
</rdf:Property>
...
```

Figure 6: Conceptual Model (RDFS)

Besides the graphical syntax for the CM, there is also an RDFS syntax as illustrated by Figure 6 (the syntax used in the actual encoding in the current software).

During the actual run-time process of presentation generation the CM is on request populated with instances (RDF statements), which represent the query result coming from the Application Layer. Note that the source data generally comes from different (heterogeneous) sources and that the actual instance retrieval is performed by the Integration Engine. Figure 7 presents a concrete example of instances adhering to our CM as a response to the given query, which propagated from the Application Layer.

```
<Painting rdf:ID="Painting_ID01">
    <name>
        <String>
             <data>The Stone Bridge</data>
        </String>
    </name>
    <year>
        <Integer>
             <data>1638</data>
        </Integer>
    </year>
    <picture>
        <Image rdf:about="http://www.rijksmuseum.nl/ariadata/image/SK/ORG/SK-A-1935.org.jpg">
             <data>On Canvas "The Stone Bridge"</data>
        </Image>
    </picture>
    <exemplifies rdf:resource="#Technique_ID01"/>
    <painted_by rdf:resource="#Painter_ID01"/>
</Painting>
...
```

Figure 7: Conceptual Model Instance (RDF)

RDFS system properties as well as some other design choices (e.g. modeling a bag of instances as a repeating property) were made to be compatible with the RDF(S) back-end of Protege 2000 [31]. Protege 2000 is a graphical tool intended for defining domain ontologies and their instances, which in our case might be used by the designer while creating the CM.

5 Application Model

The Application Model (AM) describes at the logical level the hypermedia aspects of the presentation. CM does not suffice to model a web application [33], one needs to define the navigational view over the CM, i.e. the AM. As mentioned in Section 3, by decoupling navigation and rendering in the presentation we can first focus on "what" is to be presented and then on "how" it is going to be presented.

The AM is based on the concept of slice [26] which represents a meaningful presentation unit. Each slice is associated to a particular concept from the CM (which acts as the slice owner). The definition of a slice is recursive: a slice can contain properties or slices which do not necessarily

belong to the slice owner concept but to different concepts[8]. In order to make such a slice embedding possible the owner concept must be related to the other concepts by relationships present in the CM (or a relationship derived from the CM by transitivity).

Slices are related to each other by slice relationships. Slice relationships represent abstractions of the presentation primitives: navigational (hyperlink), space and time relationships [19] between slices. Note that, as stated in Section 4, the CM relationships between two concepts can have cardinality one-one (single) or one-many (multiple). We distinguish two types of slice relationships:

- Aggregation relationships which are simple slice insertions when the owner concepts have a one-one relationship or access structures (index, tour, indexed guided tour, etc.) when the owner concepts have a one-many relationship.
- Reference relationships which are relationships that preferably get materialized with hyperlinks in the PM.

It is the designer's task to carefully specify slice relationships taking in account the relationships from the CM. Figure 8 presents a part of the AM for our museum example.

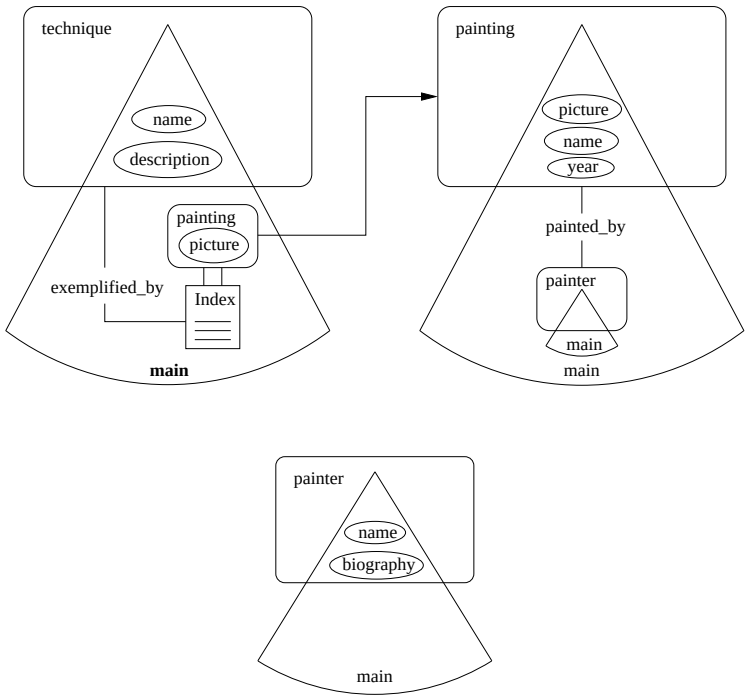


Figure 8: Application Model

The AM example is composed from three slices owned by three concepts: *technique*, *painting*, and *painter*. Slice names are prefixed with the name of the owning concept (separated by '.'). All three slices serve as entry points in the presentation (denoted by the use of *main*). The first slice, for technique, contains the *name*, the *description*, and the *pictures* of the paintings that exemplify the particular painting technique. A picture is linked with a reference relationship to

[8]Due to their nested nature slices are also called M-slices where 'M' stands for Matrejeska, the Russian doll [23].

a detailed description of the painting captured in the second slice. The description of a painting contains the actual *picture*, its *name*, the *year* when it was painted, as well as the information about the painter from the bottom slice. The painter slice presents the *name* and the *biography* of the painter.

In order to represent the AM in RDF it is necessary to define RDFS constructs for that purpose. We introduce the RDFS primitives *Slice* and *Link* classes, in Figure 9. The *source* and *destination* properties specify the source and destination of a link. The source is of type *Anchor*, which is a superclass of the *Media* and *Slice* types. This enables an anchor to be a full slice or just a media item in a slice. The destination is of type *Slice*. These constructs are defined in Figure 9 and are used in the AM specification of our museum example presented in Figure 10.

```
<rdfs:Class rdf:ID="Slice"/>
<rdfs:Class rdf:ID="Anchor"/>
<rdfs:Class rdf:ID="Link"/>
<rdf:Description rdf:about="#Slice"/>
    <rdfs:subClassOf resource="#Anchor">
</rdf:Description>
<rdf:Description rdf:about="#Media"/>
    <rdfs:subClassOf resource="#Anchor">
</rdf:Description>
<rdf:Property rdf:ID="source">
    <rdfs:domain rdf:resource="#Link"/>
    <rdfs:range rdf:resource="#Class"/>
</rdf:Property>
<rdf:Property rdf:ID="destination">
    <rdfs:domain rdf:resource="#Link"/>
    <rdfs:range rdf:resource="#Slice"/>
</rdf:Property>
```

Figure 9: Slice Model (RDFS)

All slice types are subclasses of the *Slice* class. They are referred as new properties attached to concepts from the CM (each with a name derived from the name of the slice that is pointed to). We distinguish the following slice properties:

- Properties that have identical names as properties of the original CM. They represent original concept attributes and relationships between concepts. Such relationships are needed when an attribute/slice belonging to a different concept is to be included in the current slice.
- Properties that have a name derived from the name of the slice that they are referring to. They represent slices that belong to the current concept.

Each slice reference (link) has its own class defined as a subclass of the class *Link*. In Figure 10 there is a link between the image of a painting (the source) and the main slice that describes this specific painting (the destination). The *source* and *destination* properties point to the appropriate resources.

```
<rdfs:Class rdf:ID="Slice.technique.main">
    <rdfs:subClassOf rdf:resource="#Slice"/>
</rdfs:Class>
<rdfs:Class rdf:ID="Slice.painting.main">
    <rdfs:subClassOf rdf:resource="#Slice"/>
</rdfs:Class>
<rdfs:Class rdf:ID="Link_exemplified_by"
           rdfs:subClassOf="#Link">
</rdfs:Class>
<rdf:Property rdf:ID="source">
    <rdfs:domain rdf:resource="#Link_exemplfied_by"/>
    <rdfs:range rdf:resource="#Image"/>
</rdf:Property>
<rdf:Property rdf:ID="destination">
    <rdfs:domain rdf:resource="#Link_exemplified_by"/>
    <rdfs:range rdf:resource="#Slice.painting.main"/>
</rdf:Property>
<rdf:Property rdf:ID="slice.painting.main">
    <rdfs:domain rdf:resource="#Painting"/>
    <rdfs:range rdf:resource="#Slice.painting.main"/>
</rdf:Property>
<rdf:Property rdf:ID="image">
    <rdfs:domain rdf:resource="#Painting"/>
    <rdfs:range rdf:resource="#Image"/>
</rdf:Property>
<rdf:Property rdf:ID="name">
    <rdfs:domain rdf:resource="#Painting"/>
    <rdfs:range rdf:resource="#String"/>
</rdf:Property>
<rdf:Property rdf:ID="year">
    <rdfs:domain rdf:resource="#Painting"/>
    <rdfs:range rdf:resource="#Integer"/>
</rdf:Property>
<rdf:Property rdf:ID="painted_by">
    <sys:cardinality rdf:resource="#single"/>
    <rdfs:domain rdf:resource="#Slice.painting.main"/>
    <rdfs:range rdf:resource="#Slice.painter.main"/>
</rdf:Property>
...
```

Figure 10: Application Model (RDFS)

Once an AM is encoded in RDFS, we can query it using RQL [1, 27]. Asking the query against the AM enables the user to decide not only about the retrieved data but also on what will be the navigational structure of the data presentation. RQL proves to be a powerful language to express uniformly queries over both RDF descriptions and schemas [27]. For instance, the query `Slice` retrieves all the slice instances that populate the AM, while `subClassOf(Slice)` provides all the slice types used in the AM. Introducing inheritance at slice level will not only better foster slice reuse in the AM but will also enable more advanced queries at schema level. RQL

has the advantage to support the subclassing mechanism in querying schemas, feature that is not present in plain XML query languages (e.g. XQuery [12]). Figure 11 provides an RQL example of the following query: "Retrieve the painting technique named "chiaroscuro"". In other words, it selects the variable S instantiations which are of type *Slice.technique.main* and have a *name* property equal to "chiaroscuro".

```
select S
from {S:Slice.technique.main}name{X}
where X = "chiaroscuro"
```

Figure 11: Query (RQL)

If we apply this query to the data modeled by the AM, we obtain an instance of the model, a part of it being shown in Figure 12. In our example, as there is only one painting technique with the "chiaroscuro" *name*, there is also only one slice instance of type *Slice.technique.main*. Note that there can be many pictures that exemplify this technique. Our example illustrates only one such picture, "The Stone Bridge" of Rembrandt van Rijn.

```
<Slice.technique.main rdf:ID="Slice.technique.main_ID1">
    <exemplified_by>
        <Link_exemplified_by>
            <source>
                <Image rdf:about="http://www.rijksmuseum.nl/ariadata/image/SK/ORG/SK-A-1935.org.jpg"/>
            </source>
            <destination rdf:resource="#Slice.painting.main_ID1"/>
        </Link_exemplified_by>
    </exemplified_by>
    ...
</Slice.technique.main>
<Slice.painting.main rdf:ID="Slice.painting.main_ID1">
    <name>
        <String>
            <data>The Stone Bridge</data>
        </String>
    </name>
    <year>
        <Integer>
            <data>1638</data>
        </Integer>
    </year>
    <image>
        <Image rdf:about="http://www.rijksmuseum.nl/ariadata/image/SK/ORG/SK-A-1935.org.jpg"/>
    </image>
    <painted_by rdf:resource="#Slice.painter.main_ID1"/>
    ...
</Slice.painting.main>
```

Figure 12: Application Model Instance (RDF)

6 User/Platform Adaptation

In the previous sections we have seen how the specification of the Conceptual Model and the Application Model contribute to the design of the target presentations. As we have mentioned earlier, the concept of adaptation [7] is an important part, that however easily complicates the design process. While this paper is not solely devoted to the adaptation issue, we do want to shortly demonstrate how a requirement to support adaptation influences the design process.

We can distinguish two kinds of adaptation in the context of our Hera methodology. First, we allow that the generation process is adapted, which means that the generation of the hypermedia presentation is dependent on a user profile (that includes e.g. user preferences and platform descriptions). Second, the hypermedia presentations that are generated can be adaptive themselves, so that these presentations can change while the user is browsing them. Now, we will address these two issues, as far as their specification aspect is concerned.

The first and most straightforward kind of adaptation can also be termed personalization (although this term is not completely accurate)[9]. It means that the generation process is based on (configured by) available information that describes the situation in which the user will use the generated presentation. By acknowledging this situation the generation process can better meet the demands of the user. We will show by an example how this notion of adaptation is supported in this methodology. In the previous section an Application Model was given, for a regular (i.e. not-personalized) generation process. When we want to specify that the generation process should take into account certain conditions on the user's situation, we have to include these conditions and their consequences in the AM. Some elements of the AM can then become optional depending for instance on the user's preferred platform as it is shown in Figure 13 for the attribute *picture* in the slice *slice.painting.main*. This attribute is guarded with a condition and will not be displayed if the user is browsing the presentation with a WAP phone.

The second kind of adaptation that we are considering in this research is that included in the adaptive hypermedia presentations that are generated. It means that the hypermedia presentations themselves change while being browsed. These changes represent that the user's browsing behavior is the basis for offering different pages or links. Usually, the rationale behind this kind of adaptation is that the user's browsing behavior indicates the user's knowledge or interest, and therefore the information provider could choose to react to changes in that knowledge or interest by guiding the user to more appropriate information [7]. Here we will concentrate on the generation of the software and data that allow adaptation to be implemented. For instance, Figure 13 shows an example of two optional subslices *slice.painter.main* and *slice.painter.main'*. These are embedded in the *slice.painting.main* slice and show the painter's name with his biography in case of the first user's visit (*slice.painter.main*) and without the biography (only with the link to the biography slice), once the user has seen the biography before (*slice.painter.main'*). While this specifies the possible choices it does not specify how the choice is made. For that we follow the lessons from the reference model from [8].

[9]Sometimes the term adaptable is used (as opposed to adaptive), but we will not make that subtle distinction here.

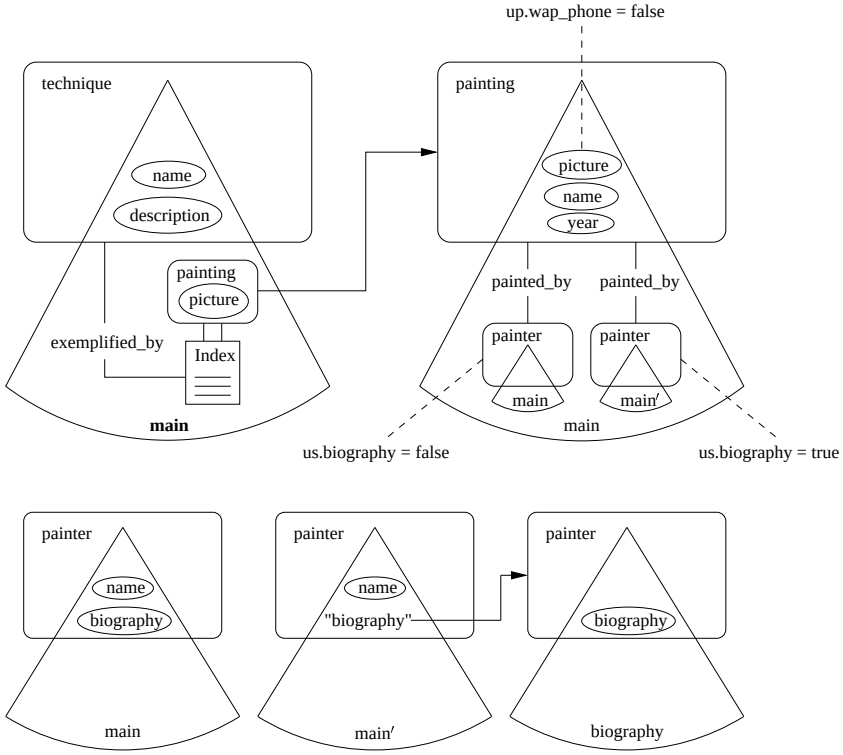


Figure 13: Application Model with User/Platform Adaptation

That reference model suggests to achieve a clear separation of the aspects that make up the adaptation. Following this approach we split the UAM into several submodels, which are discussed below.

- The Domain Model (DM) represents the domain of objects that are relevant in the adaptation process. In our framework the domain model consists of elements from CM and AM.
- The User Model (UM) represents the user's preferences and his browsing sequence, i.e. there is a (possibly) different UM for every user in the system. The UM consists of two parts (1) a user profile (UP), which represents a static[10] set of user settings (e.g. visual preferences, used platform etc.) and (2) a user session (US) representing the dynamic user's state within one session (e.g. whether or not a slice instance has been seen etc.).

  The UP is a table, which associates a fixed set of properties (of interest w.r.t. the adaptation), e.g. *wap_phone*, with Boolean values, in this case indicating presence or absence.

  The US is a table, which associates objects from the DM, e.g. *biography*, with Boolean values, indicating whether the objects were seen or not. The US unlike the UP is updated during the session and dynamically changes its content.
- The Adaptation Model (AdM) is based on the UM and UP; it describes how the next data delivery is determined and also how the UM is to be updated accordingly. Note that in general, in conformance with [8], the adaptation relies on the existence of some

[10]Static with respect to one session.

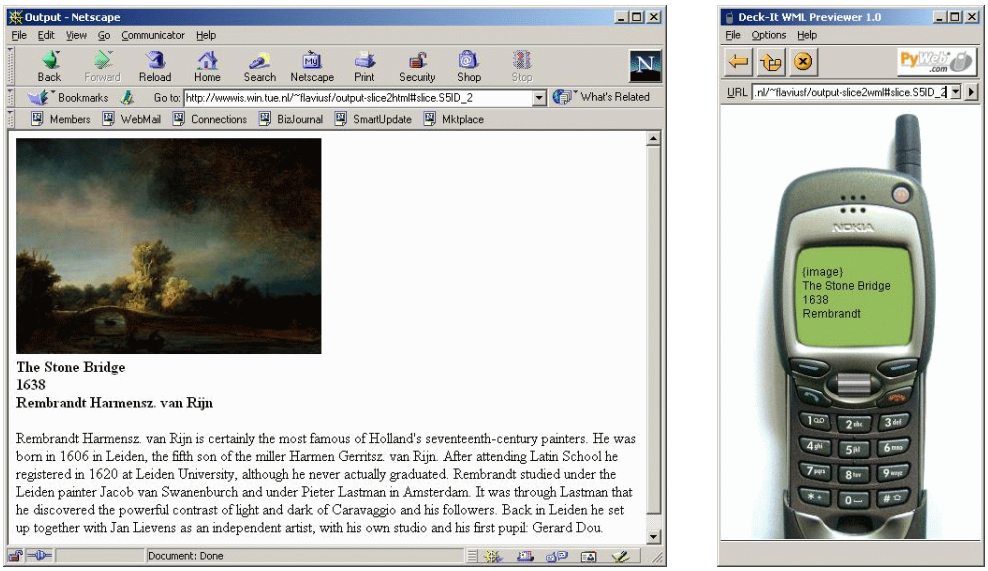
Adaptation Engine, a piece of standard software that is actually capable of performing the adaptation and that interfaces with the presentation platform, e.g. the browser. However, the Adaptation Engine needs to be configured for the application, and that configuration is in our case specified in the AdM. The AdM is expressed in terms of rules. We distinguish (1) application rules and (2) update rules.

Application rules describe the behavior of the application; typically these are rules introduced in the AM as conditional attributes or slices, as depicted in Figure 13. They are executed by the Application Engine, while (re)generating the application.

Update rules [6] are responsible for keeping the UM up-to-date by changing the relevant values in the US table. Such a rule typically consists of an event (e.g. a request for the slice *slice.painter.main*) that triggers an action (e.g. update *us.biography:= false*). These rules can be composed in a recursive manner triggering each other[11]. The update rules are executed by the Adaptation Engine, which in our case is the AHA software package [12].

7 Rendering

While in the previous phase the AM is constructed, possible including adaptation aspects, in order to view such an AM instance one needs to build code generators specific to a user's platform. Two code generators were developed: one outputs HTML [2] code (for PC Web browsers) and the other WML [18] code (for WAP phone browsers). Figure 14 presents the snapshots of the presentation rendering for the two platforms.



HTML rendering
(Netscape Navigator−Web Browser)

WML rendering
(Nokia 7110−Wap Phone Emulator)

Figure 14: Rendering for Different Platforms

[11] Here, we assume that the designer provides a correct specification, i.e. we assume confluence and termination of the specified rules.

[12] Details about the AHA software are available from `http://aha.win.tue.nl`.

The code generator is an XSLT [13] processor that converts an input XML [9] file into a targeted XML[13] file based on an XSLT specification (stylesheet). The AM instances are stored in an XML file that serializes their RDF representations. By applying the transformation described in the stylesheet one generates code for the desired platform. Figure 15 provides an excerpt from a transformation stylesheet used to generate HTML code.

```
<xsl:template match="/RDF">
    <HTML>
    <HEAD>
    <TITLE>Output</TITLE>
    </HEAD>
    <BODY>
        <!-- each slice has an HTML TABLE implementation -->
        <xsl:apply-templates select="*">
    </BODY>
    </HTML>
<xsl:template>
```

Figure 15: HTML Code Generator (XSLT)

As a translation scheme we chose to implement every slice as a TABLE in HTML and as a CARD in WML. Slice contents are TABLE rows in HTML and paragraphs in WML. Each slice is identified by an anchor in HTML and by a CARD id in WML.

We plan to extend this to also generate input code for the Cuypers [38] engine developed at CWI, Amsterdam, that provides a fine-grain customizable presentation in SMIL [14], a multimedia markup language supported among others by Internet Explorer. As an alternative, one can build a code generator based on the Presentation Model [19] developed in the Presentation Design step, the intermediate step between Application Design and Rendering.

8 Conclusions and Future Work

This paper has concentrated on data-driven Web applications that automatically generate hypermedia presentations for the Semantic Web. We have addressed the need for a framework that supports the engineering of such applications. We have shown the main aspects of our Hera framework, and the different design phases. By separating (1) the conceptual (or semantical) description of data, (2) the navigational aspects of its hypermedia presentation and (3) the rendering of the presentation, the process of the design of the application improves significantly. The different specifications (models) have been illustrated using the running (museum) example. Within the context of the Application Design we have also addressed the issue of adaptation, in terms of personalization (to reflect e.g. user preferences or platform used), as well as adaptation within the generated presentation (generating adaptive hypermedia).

The future work includes further developments to facilitate (user-)adaptation at all levels of

[13] In fact the XML requirements can be relaxed to markup languages that have a not so strict grammar, e.g. HTML.

the design process. Currently, while part of the Hera project is involved with data integration (the phase that "fills" the data repository), the research on adaptation is not strongly linked to that data integration. We plan to extend the support for adaptation to the phase of data integration.

At the same time we are involved in the further development of tools to execute the presentation rendering. While an interface with the Cuypers engine offers a partial solution, we are also working on an autonomous tool that can generate the different presentations based on the rendering specifications.

The Hera design framework concentrates currently on the specification of the different aspects of the hypermedia presentation to be generated. The existing software tools that accompany the design framework make it possible to interpret the specifications and generate the appropriate hypermedia presentation. The future work mentioned above fall exactly in that category. However, in order to facilitate the designers of such applications there is also a strong need for tools that help during the specification phases. These tools can be seen as authoring tools. They help to construct the different parts of the specifications, without having to have a very advanced understanding and experience in that specification technique. By developing (on a research prototype basis) authoring tools to construct the conceptual and application diagrams we could offer more accessible user-interfaces for the activities in these design phases.

Furthermore, we need to further develop tools that specially help to specify the aspect of adaptation. As shown in [8] a separation of concerns is essential for a good authoring process with respect to adaptation: splitting domain model, user model, and adaptation rules enables just this. In connection with the AHA project, that successfully builds on the ideas from [8], we try to develop authoring tools specifically for the purpose of adaptation.

9 Acknowledgements

The authors would like to thank Rijksmuseum [14] in Amsterdam, for their kind permission to use copyrighted data in our example. We also wish to thank our colleagues in the RTIPA and Dynamo projects, specially Jacco van Ossenbruggen and Lynda Hardman from CWI, Amsterdam, for helpful discussions.

References

[1] Sofia Alexaki, Vassilis Christophides, Greg Karvounarakis, and Dimitris Plexousakis. The RDFSuite: Managing Voluminous RDF Description Bases. In *Second International Workshop on the Semantic Web (SemWeb '01), WWW10*, 2001.

[2] Murray Altheim and Shane McCarron. XHTML 1.1 - Module-based XHTML. Technical report, W3C, 2001. Available from `http://www.w3.org/TR/xhtml11`.

[14] The Rijksmuseum Web-site is available from `http://www.rijksmuseum.nl`.

[3] Venkatraman Balasubramanian, Michael Bieber, and Thomas Isakowitz. A case study in systematic hypermedia design. *Information Systems*, 26(4):295–320, 2001.

[4] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001. Available from `http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html`.

[5] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes. Technical report, W3C, 2001. Available from `http://www.w3.org/TR/xmlschema-2`.

[6] Paul De Bra, Ad Aerts, Geert-Jan Houben, and Hongjing Wu. Making General Purpose Adaptive Hypermedia Work. In *WebNet 2000 World Conference on the WWW and Internet*. AACE, 2000.

[7] Paul De Bra, Peter Brusilovsky, and Geert-Jan Houben. Adaptive Hypermedia: From Systems to Framework. *ACM Computing Surveys*, 31(4es), 1999.

[8] Paul De Bra, Geert-Jan Houben, and Hongijng Wu. AHAM: A Dexter-based Reference Model for Adaptive Hypermedia. In *The 10th ACM Conference on Hypertext and Hypermedia (Hypertext '99)*. ACM, 1999.

[9] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible Markup Language (XML) 1.0 (Second Edition). Technical report, W3C, 2000. Available from `http://www.w3.org/TR/2000/REC-xml-20001006`.

[10] Dan Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. Technical report, W3C, 2000. Available from `http://www.w3.org/TR/2000/CR-rdf-schema-20000327`.

[11] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web Modeling Language (WebML): a Modeling Language for Designing Web Sites. In *Nineth International World Wide Web Conference (WWW9)*. Elsevier, 2000.

[12] Don Chamberlin, James Clark, Daniela Florescu, Jonathan Robie, Jerome Simeon, and Mugur Stefanescu. XQuery 1.0: An XML Query Language. Technical report, W3C, 2000. Available from `http://www.w3.org/TR/xquery`.

[13] James Clark. XSL Transformations (XSLT) Version 1.0. Technical report, W3C, 1999. Available from `http://www.w3.org/TR/xslt`.

[14] Aaron Cohen. Synchronized Multimedia Integration Language (SMIL 2.0) Specification. Technical report, W3C, 2001. Available from `http://www.w3.org/TR/smil20`.

[15] David C. Fallside. XML Schema Part 0: Primer. Technical report, W3C, 2001. Available from `http://www.w3.org/TR/xmlschema-0`.

[16] Dieter Fensel, Ian Horrocks, Frank van Harmelen, Deborah L. McGuinness, and Peter F. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.

[17] Adam Field, Pieter Hartel, and Wim Mooij. Personal DJ, an Architecture for Personalized Content Delivery. In *Tenth International World Wide Web Conference (WWW10)*. ACM, 2001.

[18] WAP Forum. Wireless Markup Language Version 2.0. Technical report, 2001. Available from `http://www1.wapforum.org/tech/documents/WAP-238-WML-20010626-p.pdf`.

[19] Flavius Frasincar, Geert-Jan Houben, and Richard Vdovjak. An RMM-Based Methodology for Hypermedia Presentation Design. In *Fifth East-European Conference on Advances in Databases and Information Systems (ADBIS '01)*. Springer, 2001.

[20] Geert-Jan Houben. HERA: Automatically Generating Hypermedia Front-Ends for Ad Hoc Data from Heterogeneous and Legacy Information Systems. In *Third International Workshop on Engineering Federated Information Systems (EFIS 2001)*. Aka and IOS Press, 2000.

[21] Jane Hunter and Carl Lagoze. Combining RDF and XML Schemas to Enhance Interoperability Between Metadata Application Profiles. In *Tenth International World Wide Web Conference (WWW10)*. ACM, 2001.

[22] Tomas Isakowitz, Michael Bieber, and Fabio Vitali. Web Information Systems. *Communications of the ACM*, 41(7):78–80, 1998.

[23] Tomas Isakowitz, Arnold Kamis, and Marios Koufaris. Extending RMM: Russian Dolls and Hypertext. In *30th Hawaii International Conference on System Sciences*. Computer Society Press, 1997.

[24] Tomas Isakowitz, Arnold Kamis, and Marios Koufaris. Reconciling Top-Down and Bottom-Up Design Approaches in RMM. In *Workshop on Information Technologies and Systems*, 1997.

[25] Tomas Isakowitz, Arnold Kamis, and Marios Koufaris. The Extended RMM Methodology for Web Publishing. Technical report, New York University, 1998. Available from `http://rmm-java.stern.nyu.edu/rmm/`.

[26] Tomas Isakowitz, Edward A. Stohr, and P. Balasubramanian. RMM: A Methodology for Structured Hypermedia Design. *Communications of the ACM*, 38(8):34–44, 1995.

[27] Gregory Karvounarakis, Vassilis Christophides, Dimitris Plexousakis, and Sofia Alexaki. Querying Community Web Portals. Submitted for publication, 2001.

[28] Reinhold Klapsing and Gustaf Neumann. Applying the Resource Description Framework to Web Engineering. In *First International Conference on Electronic Commerce and Web Technologies (EC-Web)*. Springer, 2000.

[29] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Technical report, W3C, 1999. Available from `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222`.

[30] San Murugesan, Yogesh Deshpande, Steve Hansen, and Athula Ginige. Web Engineering: A New Discipline for Web-Based System Development. In *First ICSE Workshop on Web Engineering, ICSE '99*. ACM, 1999.

[31] Natalya F. Noy, Michael Sintek, Stefan Decker, Monica Crubezy, Ray W. Fergerson, and Mark A. Musen. Creating Semantic Web Contents with Protege-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.

[32] Gustavo Rossi, Daniel Schwabe, and Robson Mattos Guimaraes. Designing Personalized Web Applications. In *Tenth International World Wide Web Conference (WWW10)*. ACM, 2001.

[33] Gustavo Rossi, Daniel Schwabe, and Fernando Lyardet. Web Application Models are more than Conceptual Models. In *Workshop on the WWW and Conceptual Modeling, ER '99*. Springer, 2001.

[34] Daniel Schwabe and Gustavo Rossi. The Object-Oriented Hypermedia Design Model. *Communications of the ACM*, 38(8):45–46, 1995.

[35] Daniel Schwabe, Gustavo Rossi, and Simone D.J. Barbosa. Systematic Hypermedia Application Design with OOHDM. In *The Seventh ACM Conference on Hypertext (Hypertext '96)*. ACM, 1996.

[36] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures. Technical report, W3C, 2001. Available from `http://www.w3.org/TR/xmlschema-1`.

[37] Frank van Harmelen, Peter F. Patel-Schneider, and Ian Horrocks. Reference description of the DAML+OIL (March 2001) ontology markup language. Technical report, DAML, 2001. Available from `http://www.daml.org/2001/03/reference.html`.

[38] Jacco van Ossenbruggen, Joost Geurts, Frank Cornelissen, Lynda Hardman, and Lloyd Rutledge. Towards second and third generation Web-based multimedia. In *Tenth International World Wide Web Conference (WWW10)*. ACM, 2001.

[39] Richard Vdovjak and Geert-Jan Houben. RDF-Based Architecture for Semantic Integration of Heterogeneous Information Sources. In *International Workshop on Information Integration on the Web (WIIW '01)*, 2001.