Hypermedia Presentation Generation in Hera

Flavius Frasincar [a,b,*] Geert-Jan Houben [a,c] Peter Barna [a]

[a] *Eindhoven University of Technology*
PO Box 513, 5600 MB Eindhoven, the Netherlands

[b] *Erasmus University Rotterdam*
PO Box 1738, 3000 DR Rotterdam, the Netherlands

[c] *Delft University of Technology*
PO Box 5031, 2600 GA Delft, the Netherlands

Abstract

Hera is a model-driven methodology for designing Semantic Web Information Systems (SWIS). Based on the principle of separation-of-concerns, Hera defines models to describe the different aspects of a SWIS. These models are represented using RDF, the foundation language of the Semantic Web. Hera is composed of two phases: the data collection phase, which integrates data from different sources, and the presentation generation phase, which builds a hypermedia presentation for the integrated data. The focus of this paper is on the hypermedia presentation generation phase and the associated model specifications. The Hera presentation generation phase has two variants: a static one that computes at once a full Web presentation, and a dynamic one that computes one-page-at-a-time by letting the user influence the next Web page to be presented. The dynamic variant proposes, in addition to the models from the static variant, new models to capture the data resulted from the user's interaction with the system. The implementation is based on a sequence of data transformations applied to the Hera models that eventually produces a hypermedia presentation.

Key words: Semantic Web Information System, RDF, hypermedia presentation generation

* Corresponding author. Tel: (+31)(10)4081340, Fax: (+31)(10)4089162
Email addresses: `flaviusf@win.tue.nl`, `frasincar@few.eur.nl` (Flavius Frasincar), `houben@win.tue.nl`, `g.j.p.m.houben@tudelft.nl` (Geert-Jan Houben), `pbarna@win.tue.nl` (Peter Barna).

1 Introduction

The World Wide Web is the most popular medium used by information systems to disseminate information to a broad audience. In 1998 the term *Web Information Systems* (WIS) was used for the first time to denote information systems that exploit the Web hypermedia paradigm for providing the application user with a user-friendly, easy-to-reach interface [1]. Some of the popular WIS are: institutional portals, community Web sites, online shops, digital libraries, e-learning courses, etc.

The opportunities that the Web offers do come at a price, there is a significant number of challenges that WIS designers need to take into account in the development of Web applications: automatic generation of the user interface, application personalization, interface portability, integration of heterogeneous data sources, etc. As traditional software engineering practices fail to cope with the Web peculiarities a new discipline called *Web engineering* aims at the establishment of systematic approaches for the development of Web applications [2].

A typical scenario in a WIS is the following: in response to a user query the system automatically generates a hypermedia presentation. The content of the hypermedia presentation is gathered from different, possibly heterogeneous, sources that are distributed over the Web. A characteristic aspect of a WIS is the *personalization* of the generated hypermedia presentation. The one-size-fits-all approach that is so typical for traditional hypermedia is not suitable for delivering information at run-time to different users with different platforms (e.g., PC, PDA, WAP phone, WebTV) and different network connections (e.g., dial-up modem, network copper cable, network fiber optic cable). The fact that the WIS needs to run on different platforms triggers the need to build different *presentations* (data arrangements on the user display) for the same set of data items. The WIS support for *user interaction* (e.g., by means of forms) enables the user to influence the generated hypermedia presentation based on his explicit needs.

Several methodologies have been proposed for the design of WIS [3]. In the plethora of proposed methodologies we distinguish the *model-driven methodologies* (e.g., WebML [4], OOHDM [5], RMM [6], UWE [7], OO-H [8], OOWS [9], OntoWebber [10]), i.e., methodologies that use models to specify the different aspects involved in the WIS design. The advantages of such model-based approaches are countless: better understanding of the system by the different stakeholders, support for reuse of previously defined models, checking the validity of the design artifacts, automatic model-driven generation of the presentation, better maintainability, etc.

The next generation Web, the *Semantic Web* [11], is an extension of the current Web in which data will have associated semantics to it. Several Semantic Web languages (e.g., RDF(S), OWL) have been proposed to represent data semantics in a uniform way at different abstractions levels. We refer to WIS that use Semantic Web technologies as Semantic Web Information Systems (SWIS).

By making data not only machine-readable but also machine-understandable the Semantic Web better supports the interoperability between different Web applications. The benefits that the Semantic Web brings to SWIS are remarkable: a large amount of annotated data accessible by any SWIS, exchange and reuse of data models between different SWIS, flexible representation of the Web semistructured data, unambiguous representation of models, inference of the intensional information stored in models, etc.

On the future Web a lot of the information will have metadata (data about the data) associated with it which will better foster the reuse and interoperability of Web applications than on the current Web. Semantic Web representations are able to cope with the semistructured aspects of data on the Web (e.g., by not enforcing the presence of a certain property on a resource). Moreover these data structures are flexible as they can be easily extended as opposed to the strict data structures of relational databases.

Differently than an XML-based model, a Semantic Web model adds more constraints to a domain specification (e.g., a unique way to represent concepts and relationships), making thus the representation less ambiguous. In addition it offers a compact representation of the data as the implicit information is computed by inference engines. Semantic Web languages have semantics grounded in Description Logics which ensures language decidability.

Model-driven SWIS design methodologies that exploit the advantages of the Semantic Web, will help the construction of successful SWIS on the future Web. The remainder of the paper is structured as follows. Section 2 presents some of the existing methodologies for designing SWIS. Section 3 introduces Hera, a SWIS design methodology that pays special attention to the modeling of the presentation personalization aspects in a SWIS. Section 4 describes the static presentation generation phase of Hera. Section 5 describes the dynamic presentation generation phase of Hera. Section 6 compares the static and dynamic presentation generation phases of Hera, and presents some the applications built using the Hera methodology. Section 7 concludes the paper and presents future work.

2 Related Work

There are few design methodologies that exploit the potential of the Semantic Web. Some of the pioneering methodologies for designing SWIS are: XWMF [12], OntoWebber [10], SEAL [13], and (A)SHDM [14,15]. Common to all these systems is the use of ontologies (as specifications of conceptualizations) [16] for describing models. These ontologies are supported by inference layers that use ontology rules (axioms) to deduce new facts based on existing facts.

After briefly describing a SWIS design methodology we will analyse how well it implements the following requirements:

- *methodology*: does the methodology provide design steps and guidelines for each step in order to produce models?
- *data integration*: does the methodology consider the integration of data coming from different heterogeneous sources?
- *personalization*: does the methodology support adaptation mechanisms in order to realize the application personalization?
- *user interaction*: does the methodology support complex forms of user interaction (e.g., by means of rich forms [1]) with the system?
- *presentation model*: does the methodology explicitly model the presentation aspects (the look-and-feel aspects, separate from navigation) of the application?

The eXtensible Web Modeling Framework (XWMF) [17] is one of the first SWIS modeling frameworks using RDF(S) [18,19]. It is based on the Web Object Composition Model (WOCM), a formal object-based language used to define the structure and content of a Web application. WOCM is a directed acyclic graph with complexons as nodes and simplexons as leaves. Complexons define the application's structure while simplexons define the application's content. Components are a special kind of complexons representing a physical entity (e.g., a Web page). The representation of an WOCM is done in RDF(S). XWMF also shows how one can reuse specifications at schema level by utilizing the RDFS subclass mechanism. XWMF doesn't provide a sequence of design steps as its focus lies solely on the models needed to specify a SWIS. XWMF offers no support for data integration, personalization, user interaction and presentation modeling.

OntoWebber [10] is an ontology-driven design methodology for building SWIS. The architecture of the system is composed of three layers: integration layer, composition layer, and generation layer. The integration layer resolves the syntactic differences between the different input data sources. With respect to this

[1] By rich forms we mean forms and their associated logic captured in scripts.

process it defines a domain ontology, as a reference ontology. The composition layer uses several ontologies (e.g., the navigation ontology, the presentation ontology, the personalization ontology, etc.) to define the site-view, a graphical representation of the instance Web site. The generation layer verifies that the constraints imposed to the SWIS are met by the instance Web site and generates the hypermedia presentation. OntoWebber uses RDF for the model representation and TRIPLE [20], an RDF query language, for querying the models. OntoWebber predefines a sequence of steps that the designer using this methodology needs to follow, and supports data integration. OntoWebber also supports the personalization and presentation modeling, but it lacks support for advanced user interaction with the system.

The Semantic PortAL (SEAL) [13,21] is a domain ontology-driven design methodology for building SWIS. SEAL proposes a number of steps for building SWIS: ontology design, data integration, site design, and implementation. In the ontology design one constructs the domain ontology which represents a uniform view over the data to be integrated. For the data integration one builds wrappers for the data sources that are mapped to the domain ontology. In the site design the navigation model, input model, and personalization model are built. The navigation model defines the navigation structure over the domain model, the input model defines forms to gather ontology data from the user, and the personalization model adapts the navigation model and input model based on the user preferences. The implementation builds a SWIS based on the previously defined models. SEAL doesn't have a well-defined methodology, but supports the data integration process. SEAL supports personalization and advanced user interaction with the system but lacks support for presentation modeling.

The Semantic Hypermedia Design Method (SHDM) [14,22] extends the expressivity of OOHDM [5] by defining ontologies for OOHDM models. These ontologies are specified in OWL [23], a more expressive language than RDFS. Similar to OOHDM, SHDM identifies four different phases: conceptual design, navigation design, abstract interface design, and implementation. For the conceptual design SHDM uses UML class diagrams which are further mapped to OWL using heuristic rules. In the navigation design one defines the navigational schema, the navigational contexts, and the access structures. The mappings between the conceptual schema and navigational schema are specified using RQL [24], an RDF query language. In SHDM it is possible to map the conceptual model corresponding to different sources to the same navigational schema. The abstract interface design defines the widget ontology to be used for defining the exhibited elements, the events sent to the system based on user actions, and the variables used to store user input data. The widgets are mapped to the navigational structures defined in the navigational schema. The implementation phase builds a SWIS based on the previous models. SHDM proposes a well-defined methodology and does to some extent support the data integration process. SHDM facilitates presentation modeling, and to some extent personalization and user interaction (as inherited from OOHDM, without using Semantic Web technologies).

The Adaptive Semantic Hypermedia Design Model (ASHDM) [15] adds personalization support to SHDM. The SHDM methodology is enlarged with new models: the adaptation context model and the adaptation model. The adaptation context model is a generalization of the user model and user context model. The user model stores domain-independent data (e.g., user's name), overlay data (e.g., knowledge of the user with respect to a certain concept), and domain-dependent data (e.g., knowledge of the user with respect to a certain domain). The user context model specifies the context characteristics that do not pertain to the user per se (e.g., the network bandwidth). The adaptation model is based on events that appear at the interaction between the user and the system, and actions that adapt the SHDM models and the user context model. An interesting feature of SHDM is meta-adaptation, i.e., choosing the right adaptation model based on the user context (e.g., the adaptation can be based on the user's learning style). (A)SHDM has a well-defined methodology and does to some degree support data integration. Personalization of the application is well sustained, and user interaction and presentation modeling are partially supported.

Table 1 shows an overview of the analysed SWIS design methodologies.

Table 1
SWIS methodologies comparison.

	XWMF	OntoWebber	SEAL	(A)SHDM
Methodology	Partial	Yes	Partial	Yes
Data integration	No	Yes	Yes	Partial
Personalization	No	Partial	No	Yes
User interaction	No	No	Partial	Partial
Presentation model	No	Partial	No	Partial

Less mature than WIS design methodologies, SWIS methodologies emphasize less the steps which are needed in order to build SWIS. They usually focus on the ontologies used for the models that specify the built applications, often failing short in providing a rigorous methodology that one needs to follow when constructing such systems. SWIS design methodologies (e.g., (A)SHDM) that extend traditional WIS design methodologies (e.g., OOHDM) inherit the original design steps providing a more structured methodology than the rest of the considered SWIS methodologies.

As the Web is hosting a large amount of data that can be reused, data integration is a topic of special interest for SWIS design methodologies. By offering the tools to describe the semantics of the data, the Semantic Web proves to be extremely useful in the Web data integration process. A common technique used by these methodologies is the wrapping of the data sources in a semantic representation. XWMF does not consider data integration as its focus lies in the presentation aspects of SWIS. Even if (A)SHDM does not have a direct mechanism to include external sources at conceptual level, it does partially support data integration by mapping the conceptual model corresponding to different sources to the same navigation model.

Most of the analysed methodologies have very little support in modeling advanced forms of user interaction with the system besides link-following. One example of advanced interaction is the use of forms for selecting data or adding new information to the system. SEAL provides a limited form of user interaction with the system by allowing the user to change the system's input data, without letting him to control the generated presentation . The only form of interaction between in (A)SHDM is the user selection of items from existing data.

A feature often neglected in the examined SWIS design methodologies is the design support for the presentation aspects (e.g., the look-and-feel aspects) of SWIS. OntoWebber considers only a few layout primitives which have limited expressive power (e.g., it is not possible to express a flow layout). Despite the increased expressivity of the (A)SHDM presentation model, to our knowledge, it is not possible to describe time-relevant aspects (e.g., slide show) which are important in hypermedia presentations on certain platforms.

3 Hera

Hera is a SWIS design methodology. The research done for developing Hera can be positioned as design science [25] as it addresses requirements that were only partially implemented by existing SWIS design methodologies. Moreover some of the solutions already proposed for the implementation of WIS requirements are done in a more effective and efficient way in Hera using Semantic Web technologies. As any design science research project the artifacts produced by Hera are constructs (vocabulary elements), models (specifications), methods (sequence of design steps), and instantiations (implementation of working systems).

Hera proposes design steps that, based on the separation-of-concerns principle, specify different aspects of a SWIS. These specification aspects are given by models that have graphical representations. Each model has a well-defined vocabulary of (graphical) primitives associated with it. The implementation of a SWIS using the Hera methodology is based on data transformations driven by Hera models. Hera has its origins in the RMM design methodology [6]. The Hera methodology is supported by the Hera Presentation Generator [26], an integrated development environment for building SWIS. This environment comprises two types of tools: tools which help the designer build models in a graphical manner, and tools which automatically generate hypermedia presentations based on previously defined design models.

3.1 Requirements

Before describing Hera we would like to emphasize the type of Web applications for which this methodology has been developed. Based on existing classifications of Web applications [2,27], the types of applications supported by Hera can be best classified as Data-Intensive Interactive Web applications. These applications are data-intensive in the sense that they use large amounts of data possibly distributed over the Web. They are also interactive, as the user is able to "control" the navigation by providing adequate input information.

The main stakeholders of the developed Web applications can be classified in two groups: customers and users, i.e., the individuals that interact with the system, and designers and developers, i.e., the team responsible for building the system. Based on these stakeholders one can identify the application requirements, which address the stakeholders' concerns. These requirements are classified as applicative (functional) requirements and managerial (non-functional) requirements.

The focus of Hera is on some of the most important, in our opinion, applicative requirements of Data-Intensive Interactive Web Applications, as data integration, interactive navigation, multi-platform access, and look-and-feel issues. The managerial aspects of these applications like security, availability, evolution, and accessibility are not the focal point of the proposed methodology. Nevertheless, we acknowledge that some of these non-functional issues like interoperability and reuse did play a role in selecting the appropriate supporting technologies. Also, the scalability and accessibility issues influenced our choice for a model-driven methodology and the related adaptation techniques.

To our knowledge none of the investigated SWIS design methodologies is able to cope in a systematic manner with the considered requirements for Data-Intensive Interactive Web applications. In the future we plan to investigate how one can extend Hera so that it is able to cope with additional applicative requirements (e.g., querying and recommendations) and managerial requirements (e.g., evolution and maintenance).

Due to the global availability of data sources for Data-Intensive Interactive Web Applications one needs a well-defined *data integration* process. The integration of Web data sources aims at providing a uniform access to multiple data sources. Such a process needs to consider the characteristics of these data sources like: autonomous, heterogeneous and overlapping, frequently changing, large size, and distributed [28].

Differently than in traditional information systems, information in the considered Web applications is explored in an explorative way rather than through "canned" interfaces [27]. One of the basic forms of interaction between the user and these applications is by means of link-following, i.e., *data navigation*, which allows the user to jump between context related information chunks. The navigation from one page to another page should support the user in fulfilling his task(s). After each "click", often, the user is one step closer to completing his task(s) and thus experiences an increasing level of satisfaction in his interaction with the system. With respect to this, it is important that the provided navigational structure supports the tasks of the users as captured in the system requirements [29].

The universal access of Data-Intensive Interactive Web applications by different users makes it difficult to have one unique interface for the interaction between the user and the system. Users have usually different goals and skills that need to be considered in the presentation generation process. Moreover, users can employ different platforms (e.g., different devices, networks) to navigate through the information which means that the presentation needs to be adapted based on the user's specific browsing context. Adapting the presentation to the user's needs is called *presentation personalization* [30].

One way in which the user can influence the next page to be be presented is by following a link. In order to further improve the communication between the user and the system, the Web applications supported by Hera are Interactive Web applications, as they allow more advanced forms of *user interaction*, e.g., the user can enter data into the system by means of forms [4]. In this way the user is able to better amend the system to his needs, improving his browsing experience.

As a competitive advantage, it has become increasingly important for Web applications to offer a pleasant look-and-feel aspect to the generated presentation. *Presentation modeling*, i.e., the layout (graphical arrangement) and style (e.g., fonts, colors) of the information in the generated presentations, plays a crucial role in achieving this purpose [31]. This is of paramount importance when WIS need to present the same information on different user's displays taking in consideration the device characteristics.

A *rigorous methodology* for designing WIS should have a well-defined sequence of steps that need to be followed by the WIS designer. The design artifacts produced at every methodological step should be based on formal descriptions involving a well-defined set of vocabulary constructs. In order to increase the utility of a WIS design methodology, there is a need for support tools that *automate* the generation of hypermedia presentations by taking in account the design models' specifications.

The construction of WIS would benefit a lot from the *reuse* of previously designed models. For example, existing domain models can be successfully recycled (possibly with some adaptations) while designing WIS. Also, WIS would profit from an open architecture architecture that fosters *interoperability*. The Semantic Web can play a key role in achieving both reuse and interoperability in WIS [32].

The previously identified requirements drive the design choices that we have made while developing Hera. As some of these requirements were only partially addressed by previous methodologies, Hera aims at providing full support for all of them in one single methodology. In this way we benefit from previous work on WIS methodologies while pushing even further the frontiers of WIS design.

3.2 Architecture

Figure 1 shows the main phases in Hera's architecture: *data collection* and *presentation generation*. The Hera methodology comes also with a straightforward implementation in which the Hera's main phases and the design steps corresponding to these phases are naturally mapped to components in a pipeline software architecture. We point out that the software based on this architecture is just one of the possible implementations of SWIS given the specifications required by the Hera methodology.

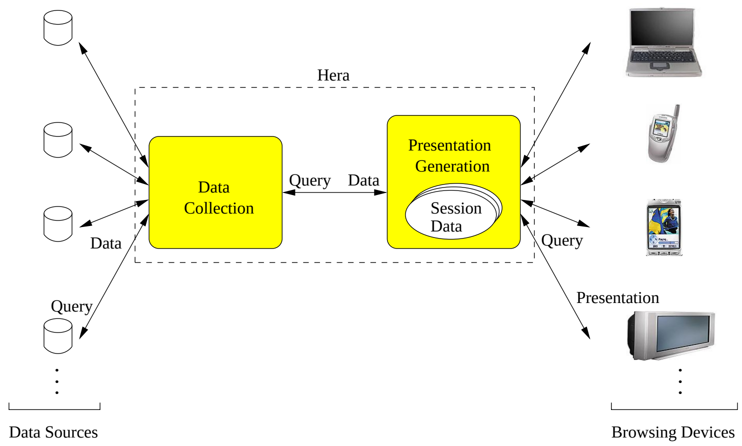

Fig. 1. Hera's main phases.

The data collection phase helps to make the data available from different sources, such that in response to a user query a data result set is obtained. In this phase of the process the integration model is defined that maps data from the different sources to a common data representation. This mapping is needed whenever for a given query the instances that compose the query result need to be retrieved. The data collection phase is outside the scope of this paper. More information on this phase can be found in [33].

The presentation generation phase builds a hypermedia presentation for the retrieved data. It is based on a sequence of data transformations driven by several models. These models depict different application aspects that are relevant in this process: what is the domain of the application, what is the navigation structure for data from this domain, how to arrange and style the data on the user's display, and how can we tailor the generated presentation based on user preferences and user browsing platform. As can be seen from Figure 1 the generated hypermedia presentations can target different platforms like PC, WAP phone, PDA, etc.

The presentation generation phase has two variants: a static one in which the user is unable to change the content of the generated hypermedia presentation and a dynamic one which considers the user interaction with the system in the process of building the next hypermedia page. In the static variant all pages are generated before the user browses the presentation and in the dynamic variant one page is generated-at-a-time during the browsing.

In order to better support the description of Hera's presentation generation phase we use a running example based on real data coming from the painting collection in a museum, the Rijksmuseum in Amsterdam. During the presentation we will show how the presentation modeling, user interaction, and personalization are supported by the Hera methodology for constructing a SWIS. The development environment that helps the designer in this process, i.e., the Hera Presentation Generator [26], is outside the scope of this paper.

3.3 RDF(S)

For the Hera specifications RDF(S) [18,19] is used. RDF(S) is the foundation language of the Semantic Web. There are several reasons for choosing RDF(S): it is flexible (it supports schema refinement and description enrichment), it is extensible (it allows the definition of new resources/properties), and it fosters Web application interoperability (it provides a framework to describe in a uniform way the data semantics). As RDF(S) doesn't impose a strict data typing mechanism it proved to be very useful in dealing with semistructured (Web) data. On top of RDF(S) high-level ontology languages (e.g., DAML+OIL [34], OWL [23]) are defined, which allow for expressing axioms and rules about the described classes giving the designer a tool with larger expressive power.

Hera models are described in RDFS. An RDFS vocabulary is developed for each model in order to define the model's concepts (which are the classes and properties to be used in a model). Differently than XML representations, RDFS models propose unambiguous and compact representation of domain specifications. Due to the flexibility of RDFS, models can be easily extended with new properties, a feature that is not easy to achieve if one chooses a relational database-based representation of models.

Model instances have an RDF representation which is validated against their corresponding schema (model). By making use of these standards we are able to reuse models between different applications. The use of RDFS allows us also to reuse existing RDFS vocabularies for expressing for example domain models or user profiles. In some applications built with Hera we successfully reused the domain model developed for museum descriptions in the TOPIA (Topic-based Interaction with Archives) project [35] and the User Agent Profile (UAProf) [36], a Composite Capability/Preference Profiles (CC/PP) [37] vocabulary for modeling device capabilities and user preferences.

4 Static Presentation Generation

In some WIS the generation of the hypermedia presentation happens before the user starts its browsing session. In such systems the interaction between the user and the system is limited to simple link-following, and Web pages are computed in advance. Examples of such WIS are: online exhibitions, Web information points (e.g., bus schedules), online tutorials, etc. The simplicity of these applications asks for a simple methodology to be used for building such systems.

The typical structure of the static variant of the presentation generation phase is given in Figure 2 in terms of three layers: the conceptual layer defines the content that is managed in the SWIS, the application layer provides the navigation structure on the data, and the presentation layer gives the presentation details that are needed for the generation of the hypermedia presentations on a concrete platform. As can be noted from Figure 2, in the static variant for the presentation generation phase the whole Web presentation is produced at once in response to a user query.

The presentation generation phase distinguishes the following steps: the conceptual design, the application design, the presentation design, and the implementation. Each design step produces appropriate models that capture the design aspects specific to this step. A model uses concepts from a model-

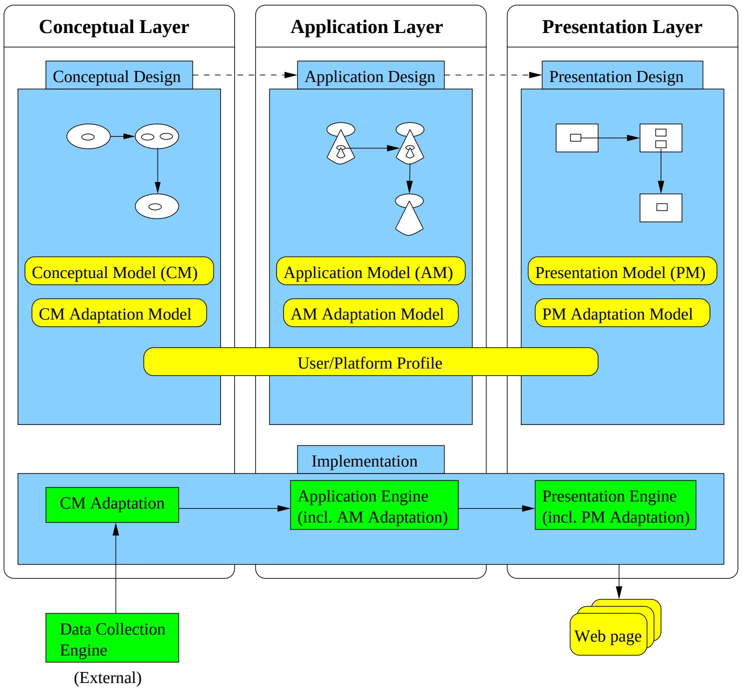

Fig. 2. Presentation generation phase (static).

specific vocabulary. In order to ease the specification of the models the model concepts have associated graphical representations. From our experience it easier for the user to grasp specifications in a graphical way than in the verbose RDF/XML serialization. In this way a model can be showed as a diagram to facilitate the development and understanding of a certain model.

Adaptation [38] is not seen as a separate design phase because this process is distributed through all the previously identified design steps. In the adaptation design the user/platform profile (UP) is defined, i.e., it is determined which are the user preferences and platform characteristics that can influence the Web presentation before the user starts the browsing session. The adaptation model specifies adaptation conditions (Boolean expressions) used to tailor the Hera models based on the UP attributes. An excerpt of the UP vocabulary is given in Figure 3. A user profile has three components (hardware, software, and preferences). Each component has properties that refer to specific values. For example, the hardware component has a property specifying the ability to display images that refers to a Boolean value.

In the user preferences component there are properties which are specific only to the domain of our application. For example the expertise level of the user strictly refers to a certain domain (the user can be a beginner in one domain and an expert in another domain). If one plans to reuse the user profile across domains, these properties need to be modeled as ternary relationships <domain, property, value> which are outside the scope of this paper.

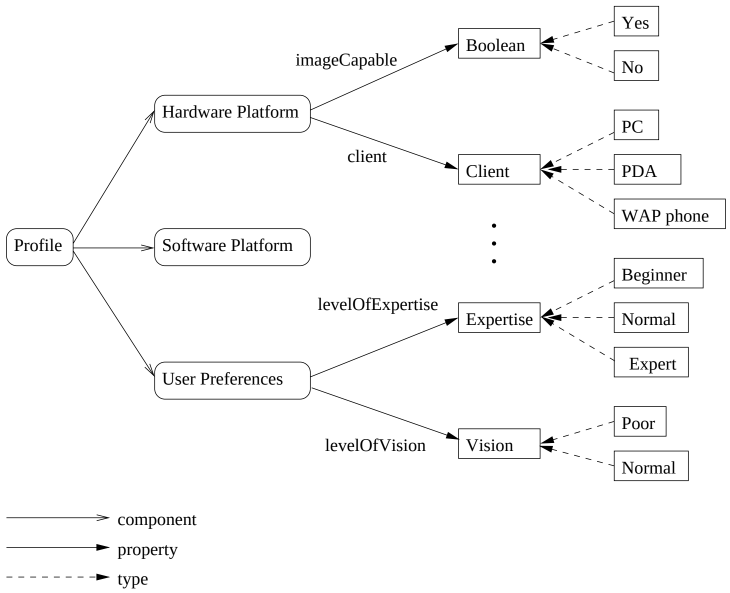

Fig. 3. User/platform vocabulary.

We present the adaptation model when we show the different design steps. If the designer is not interested in adapting the system he can ignore the adaptation aspects in the proposed methodological steps. The adaptation presented here is a fine-grained adaptation. A coarse-grained adaptation is achieved by using group profiles, instead of UPs. In this approach users with similar characteristics are assigned a group profile. One of the advantages of coarse-level adaptation is the decrease in the system's workload, as the performed adaptation is relevant for several users.

4.1 Conceptual Design

The conceptual design specifies the input data in a uniform manner, independent from the input sources. The result of this activity is the *conceptual model* (CM). From a database point of view, the CM defines the schema for the data that needs to be presented. The CM serves also as the interface between the data collection phase and the presentation generation phase of the Hera methodology.

Figure 4 shows the CM vocabulary. It defines the following notions: *concept*, *concept attribute*, and *concept relationship*. A concept represents a certain entity in a particular application domain. Concept attributes and concept relationships refer to media types and other concepts, respectively, in order to describe the properties that characterize a concept. As CM vocabulary we did use the standard RDFS concepts with three extensions: one for modeling the *cardinality* of the concept relationships, one for representing the *inverse* of the concept relationships, and one for depicting the media types, the so-called *media vocabulary*. Similar to database modeling, many-to-many concept relationships are decomposed into two one-to-many concept relationships. In this way we have only two types of cardinalities: one-to-one and one-to-many.

One can note the use of *concept relationship* in Figure 4 as a concept property (top part of figure) and as a as a standalone concept (bottom part of figure). In this representation we have adopted the property-centric view of RDF, in which one can attach properties (e.g., cardinalities) to properties (e.g., concept relationships).

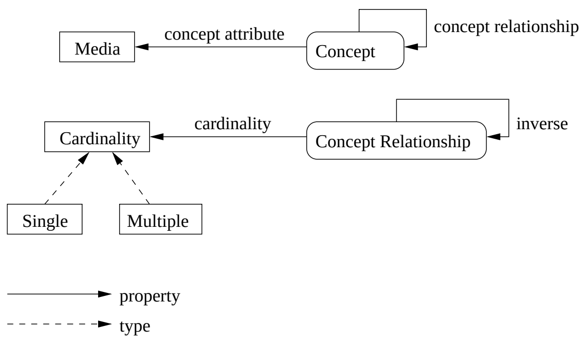

Fig. 4. Conceptual model vocabulary.

Figure 5 shows the type hierarchy in the media vocabulary. In the same way as AMACONT [39], we base our media vocabulary on a subset of the MPEG-7 standard [40]. The basic media types are: *Text*, *Image*, *Audio*, and *Video*. The figure also shows the attributes of the media types, for example the *nrChars* of a text or the *width* and *height* of an image. For the refinement of the *Text* media types the XML Schema Datatypes (e.g., *String* or *Integer*) are used (not shown in the figure). In order to achieve a high degree of application interoperability, Hera uses as much as possible the existing Web standards.

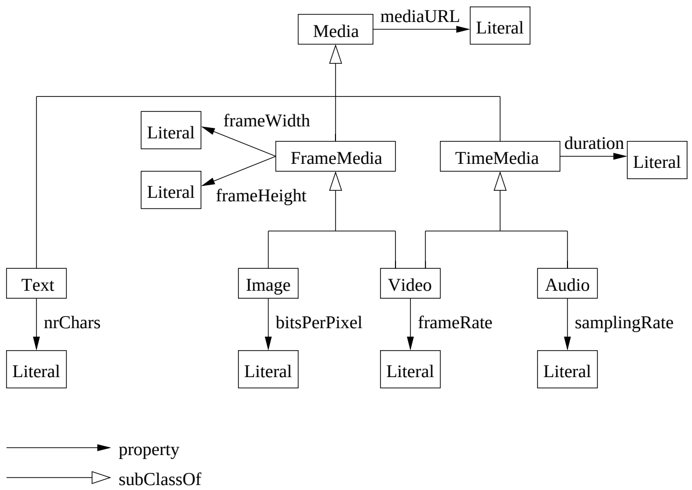

Fig. 5. Media vocabulary.

Media adaptation selects the most appropriate media items for the technical system parameters provided by different network environments and client devices. Figure 6 shows a few media adaptation examples. For devices that are not able to display images (like certain WAP phones), the images are removed from the presentation. Based on display size, large strings and images are selected for PC, and small versions of the same strings and images are selected for PDA.

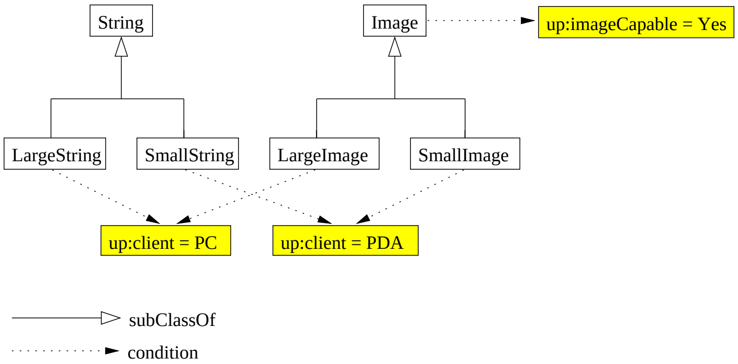

Fig. 6. Media adaptation.

Figure 7 shows an excerpt of the CM for the running example. Concepts are represented as ovals and media types as rectangles. There are three concepts: *Technique*, *Artifact*, and *Creator*. A *Creator* has two concept attributes attached to it, *cname*, for the creator's name, and *biography* for the creator's biography, both depicted by *String* items. A *Creator* is associated using the concept relationship *creates* to an *Artifact*. The cardinality of this concept relationship is one-to-many, i.e., one creator creates many artifacts. The inverse of the *creates* concept relationship is the *created_by* concept relationship. Note that both concept relationships and concept attributes are denoted as concept properties.

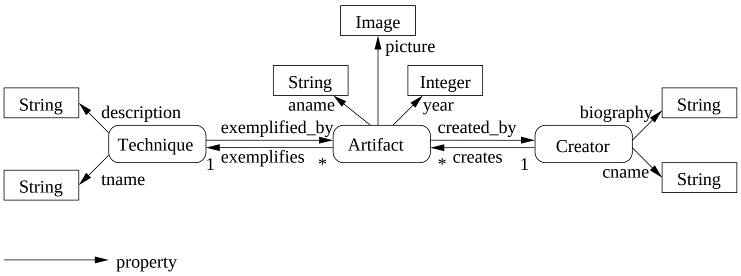

Fig. 7. Conceptual model.

The conceptual model presented in Figure 7 depicting any creator, artifact, or technique can be refined to a specific artistic domain. Figure 8 shows the specialization (in a type hierarchy) of the previous conceptual model to the painting domain. Concepts are specialized by the *subClassOf* property and concept relationships are specialized by the *subPropertyOf* property. For example, the *Creator* is specialized as a *Painter* and the *creates* relationship is specialized as *paints*.

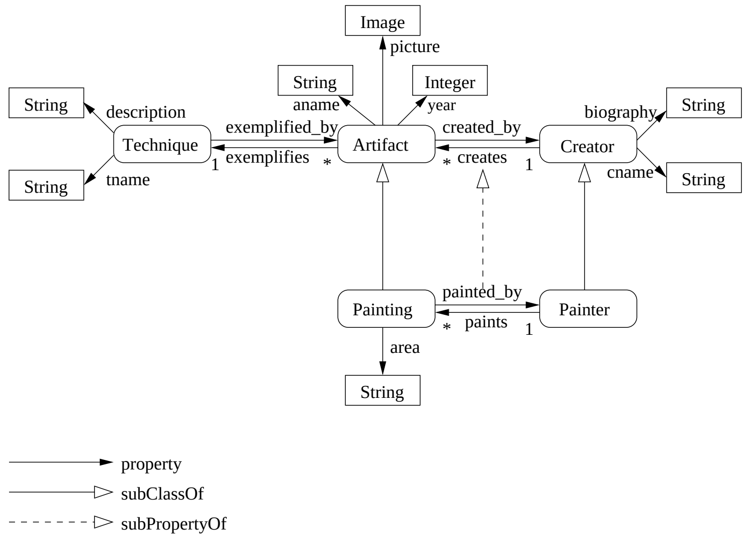

Fig. 8. Specialization in the conceptual model.

CM adaptation selects concepts or concept attributes from the CM to be used in the presentation. Figure 9 shows an adaptation example in the conceptual model. In this example the *description* of the painting technique is removed from the whole presentation if the user is not an *Expert*. This is the so-called *context-independent adaptation*, i.e., adaptation that affects the entire presentation. An example of *context-dependent adaptation*, i.e., adaptation that affects only a certain situation in a presentation, is provided in the next section.

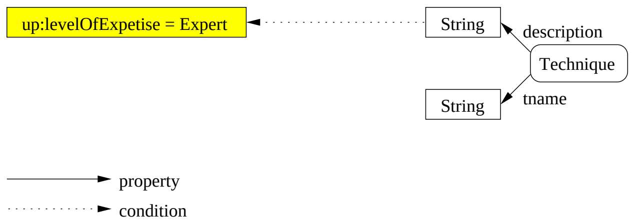

Fig. 9. Adaptation in the conceptual model.

4.2 Application Design

The application design defines the navigational aspects of the presentation to be generated. The user of a WIS needs to be able to navigate through the provided information. As CM does not suffice to model these navigational aspects of the presentation [41]: one needs to define the navigational view over the CM. The result of this activity is the *application model* (AM). From a database point of view, the AM is a view over the CM extended with navigation primitives.

Figure 10 shows the AM vocabulary. It defines the following notions: *slice*, *slice attribute*, and *slice relationship*. A slice [1] is a meaningful presentation unit that fulfills a certain communication purpose. Slice attributes are used to refer to media types. There are two types of slice relationships, *slice aggregation* and *slice navigation*. The first type of slice relationship facilitates the inclusion of a slice into another slice and the second type of slice relationship is used to define navigation between slices.

An *empty slice* [2] is a slice that has its content defined at design-time (the content can be predefined or computed by a function of other known values). Such a slice has only one attribute that refers to a media type added at design-time. A *non-empty slice* has its content defined at run-time. In order to know from where the content is to be extracted at run-time slices have associated to them an *owner* concept from CM. The *owner* attribute for an empty slice can be any concept, as the slice content is defined at design-time.

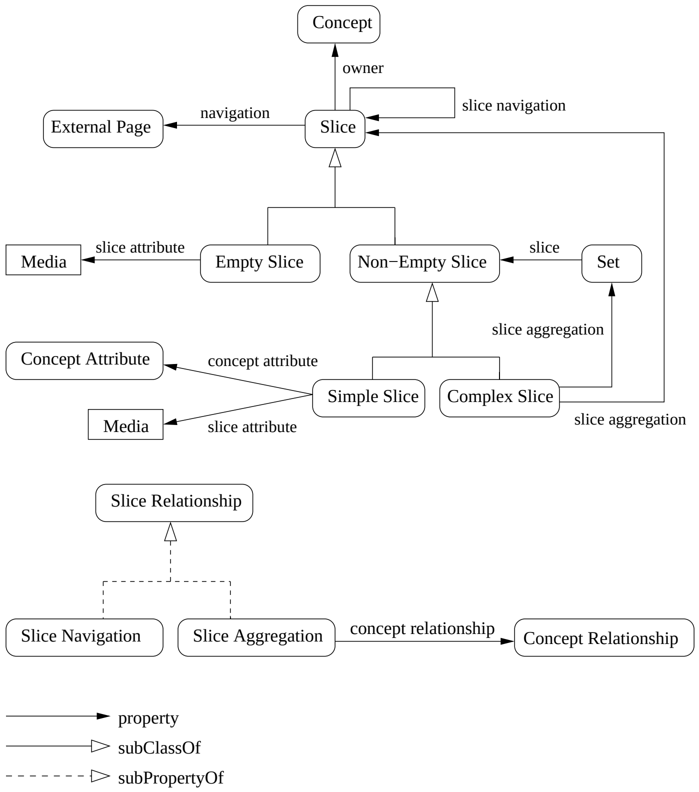

Fig. 10. Application model vocabulary.

The definition of a non-empty slice is recursive: a slice can be a *simple slice* or can contain other slices. [3] A simple slice has only one slice attribute that

[2] Dealing with data-intensive applications, by 'empty' is meant that there is no content that will populate this type of slice at run-time.
[3] Due to their nested nature, slices are also called M-slices where 'M' stands for

refers to the same media as the *concept attribute* of the owner concept from CM. A slice that aggregates other slices is called a *complex slice*. The recursion is defined by utilizing the *slice aggregation* relationship. The aggregation relationship between two slices that have two different owners needs to specify the *concept relationship* (or a relationship derived from the CM by relationship chaining) between the two owner concepts from the CM that made such an embedding possible. In case that the cardinality of this concept relationship is one-to-many the *Set* construct needs to be used. A *top-level slice* corresponds to a Web page while other slices are components of higher-level slices. Using a slice navigation relationship, a slice (the anchor) can be linked to a top-level slice. Additionally a slice can be linked to an external Web page.

In [42] is presented a more advanced way of navigating between slices in which the destination of a navigational link can be a component slice. In this case the source slice of the link can be replaced by the destination slice in the same top-level slice. This feature can be easily implemented using AJAX technology when only certain parts of a top-level slice are updated as a response to following a navigation link.

Figure 11 shows an excerpt of the AM for the running example. Slices are depicted (as their name suggests) by pizza-slice shapes. There are two slices, the main slice owned by *Technique* and the main slice owned by *Artifact*. We use the convention to denote the slice (long) name by *Slice.<concept name>.<slice short name>*, in order to distinguish them from concept names or slices with the same short name but owned by different concepts. The name of the slice owned by *Technique* is thus *Slice.Technique.main*. The slice *Slice.Technique.main* aggregates (by means of slice aggregation relationships) two simple slices and one complex slice. The simple slices *Slice.Technique.tname* and *Slice.Technique.description* are owned by *Technique*. The complex slice that aggregates *Slice.Artifact.picture* is owned by a different concept, i.e., *Artifact*. The aggregation relationship used for this embedding refers to the *exemplified_by* concept relationship between *Technique* and *Artifact*. As the cardinality of *exemplified_by* is one-to-many the *Set* construct is also inserted. In a similar manner the slice *Slice.Artifact.main* is defined. As *created_by* has cardinality many-to-one (inverse of *creates*), the *Set* construct is not used in this case. The slice navigation relationship connects the picture of an artifact *Slice.Artifact.picture* with the slice giving detailed information about that artifact *Slice.Artifact.main*.

The AM presented in Figure 11 depicting the main slices for techniques and artifacts can be refined to a specific artistic domain. Figure 12 shows the specialization (in a type hierarchy) of the previous AM to the painting domain. Slices are specialized by the *subClassOf* property. For example,

---

Matryoshka, the Russian doll [6].

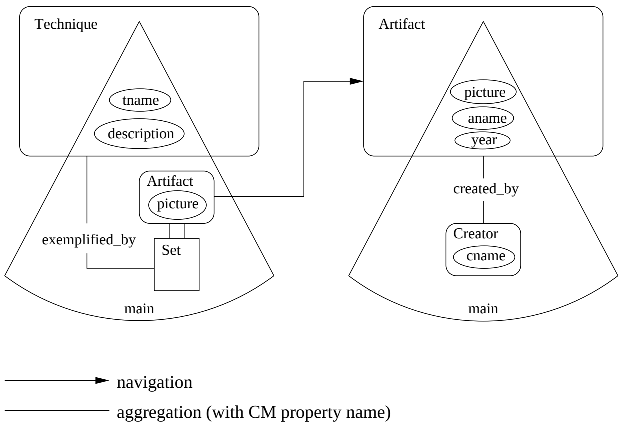

Fig. 11. Application model.

the slice *Slice.Creator.main* is specialized by the slice *Slice.Painting.main*. *Slice.Painting.main* inherits all the slice relationships of *Slice.Technique.main* and adds three new slice relationships to it: two slice aggregations and one slice navigation. The aggregation relationships refer to the slice *Slice.Technique.area* and *Slice.Technique.tname*. The navigation relationship links backwards the *Slice.Technique.tname* with the *Slice.Technique.main*.

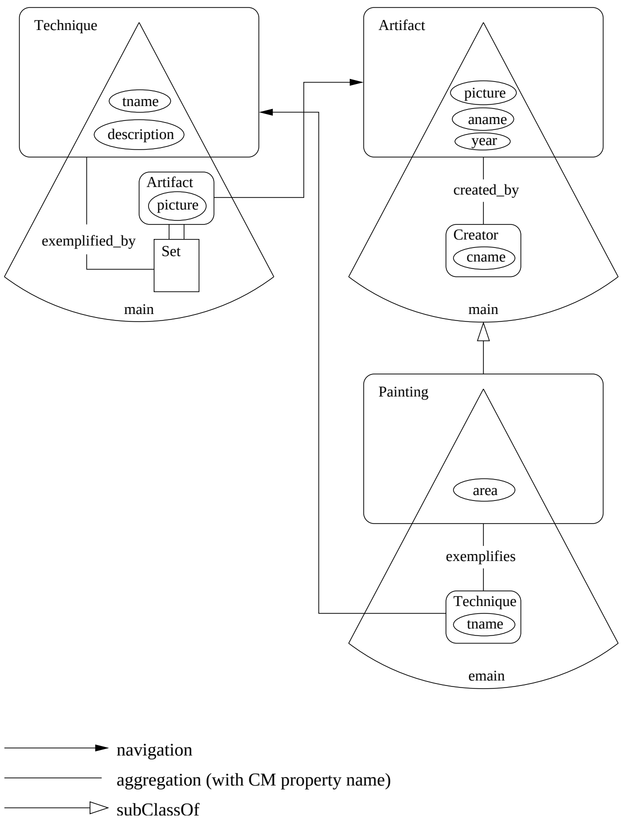

Fig. 12. Specialization in the application model.

The AM adaptation [43] is based on two typical adaptation mechanisms: *conditional inclusion of fragments* (fragments are slices in our context) and *link* *hiding* [44] (links are slice navigation relationships in our context). A link is hidden when its destination slice has a condition that evaluates to false.

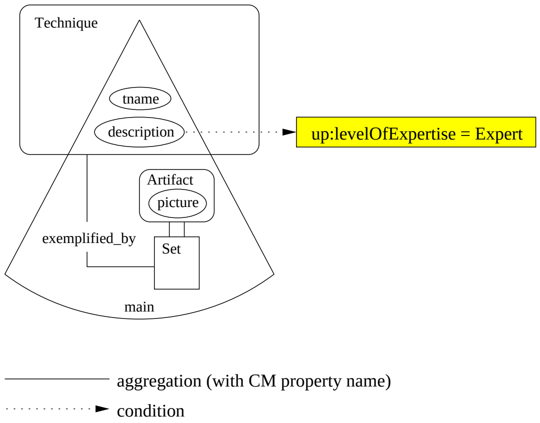

Fig. 13. Adaptation in the application model.

Figure 13 shows an adaptation example in the AM. In this example the *description* of the painting technique is removed from the main slice of this technique if the user is not an *Expert*. Later on in the presentation, the description of the painting technique can appear also for users that are not *Expert*s (at that point in the presentation, the system can consider that the user is now ready to digest more advanced information). This is the so-called *context-dependent adaptation*, i.e., adaptation that affects only the current slice (by current slice is meant the top-level slice that contains the slice with the condition). Slices that have attached conditions outside the scope of a container slice have a *context-independent adaptation*, i.e., these slices will be removed from the whole presentation, no matter where they appear. This is similar to the *context-independent adaptation* for conceptual model adaptation showed in Figure 9. Note that the removal of a concept or concept attribute from a presentation has as a consequence the removal of all associated slices (i.e., slices for which the concept is an owner) or of the slice that refers to that concept attribute, respectively.

4.3 Presentation Design

The presentation design specifies the look-and-feel aspects of the presentation to be generated, independent from the presentation implementation. The result of this activity is the *presentation model* (PM). It describes the layout and style information of the presentation. Both aspects are not to be neglected because they might have an immediate impact on the user choice for a certain application among applications offering similar functionality. The supported platforms are HTML browsers for PC, cHTML browsers for PDA, and WML browsers for WAP phones.

Figure 14 shows the PM vocabulary. It defines the following notions: *region*, *region attribute*, and *region relationship*. A region is an abstraction for a rectangular part of the display area where the content of a slice will be displayed. Each region is associated to a slice, the so-called region *owner*, from which the region content will be derived. The definition of region is very similar to that of a slice with a few simplifications and some additions. Region attributes are used to refer to media types. There are two types of region relationships, *region aggregation* and *region navigation*. The first type of region relationship facilitates the inclusion of a region into another region and the second type of region relationship is used to define navigation between regions. The classification empty/non-empty does not apply for regions as regions get their content from the slice owner always at run-time.

Differently than other modeling approaches we decided to use aggregation and navigational relationships not only in the application model but also in the presentation model. In this way the presentation model captures all the information needed to generate a hypermedia presentation. Because of this it is easier to specify the transformation of the presentation model in a hypermedia presentation. Nevertheless, this design choice comes at a price, as one needs to maintain the consistency between the application model and the presentation model.

The definition of regions is recursive: a region can be a simple region or can contain other regions. A *simple region* has only one region attribute that refers to the same media as the slice attribute of the corresponding simple slice from AM. Differently than for slices, one doesn't need to specify a corresponding concept attribute. A region that aggregates other regions is called a *complex region*. The recursion is defined by utilizing the region aggregation relationship. Another difference from slices is that for aggregation relationships there is no need to specify concept relationships. The *Set* construct, aggregation, and navigation relationships are copied for a region from the corresponding (by the owner relationship) slice. A *top-level region* corresponds to a Web page and is owned by a top-level slice.

A region has a particular *layout manager* and *style* associated with it. There are four abstract layout managers: *BoxLayout*, *TableLayout*, *FlowLayout*, and *TimeLayout*. The layout managers describe the spatial/temporal arrangements of regions embedded into another region. The list of layout managers can be easily extended with other layouts like *BorderLayout*, *OverlayLayout*, *GuidedTourLayout*, etc.

In [42] is presented an alternative way of defining layouts by using qualitative and quantitative constraints for regions. These constraints are associated to region relationships which are further classified as temporal, navigational, and spatial. Temporal relationships express the notion of time, navigational rela-

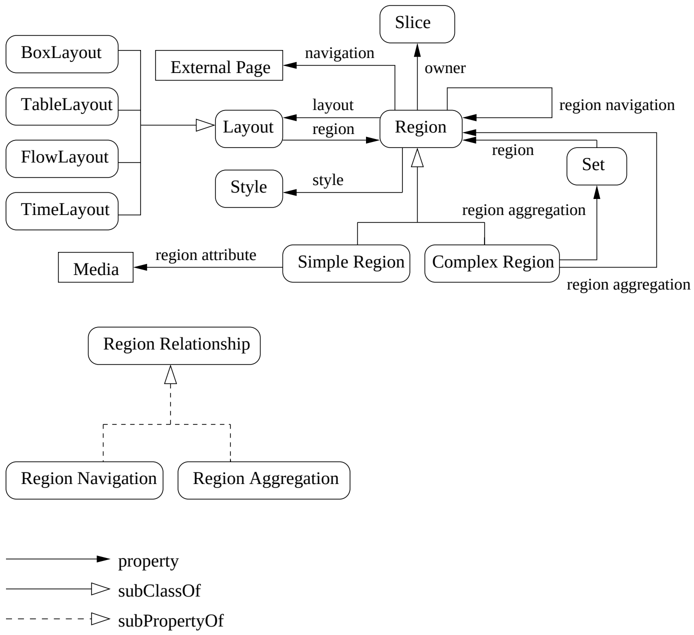

Fig. 14. Presentation model vocabulary.

tionships represent (hyper)links, and spatial relationships define the spatial arrangements in presentations.

The layout managers were inspired from the abstract user interface (XML) representations from AMACONT [31], UIML, and XIML [45]. Our decision is based on the fact that these layouts offer a detailed abstract representation of the possible spatial arrangements of the information for different user displays. In this way we are able to deploy the same WIS on a multitude of channels ensuring thus the ubiquity of our application on the Web. Note that because regions can be aggregated, layouts can also be aggregated (by means of regions), and thus one is able to build complex layouts.

The style information describes the colors, fonts, backgrounds to be used in a region, etc. Regions that do not have explicitly associated style information inherit the style of their container. In this way the designer is not forced to specify style information if that is not necessary.

The BoxLayout arranges the inner regions on one row or one column. Table 2 summarizes the possible attributes of the BoxLayout. The *height*, *width*, *border*, and *space* attributes have integer values that represent number of pixels.

TableLayout arranges the inner regions in a table. Though it can be realized by nested BoxLayouts, we implemented it separately because SWISs often present dynamically retrieved sets of data in a tabular way. Table 3 summarizes the possible attributes of the TableLayout. Due to the dynamic nature of SWIS applications, the number of items in a complex region that uses the *Set*

Table 2
BoxLayout attributes.

Attribute	Meaning	Usage	Values
axis	orientation of the layout	required	"x"|"y"
rows	number of rows	optional	integer
columns	number of columns	optional	integer
height	height of the layout	optional	integer|percentage
width	width of the layout	optional	integer|percentage
border	size of the layout border	optional	integer
space	space between content and border	optional	integer

construct is not known at design-time. In such cases one should use only one of the dimensions: *rows* or *columns*. The missing dimension is automatically computed at run-time.

Table 3
TableLayout attributes.

Attribute	Meaning	Usage	Values
rows	number of rows	optional	integer
columns	number of columns	optional	integer
height	height of the layout	optional	integer|percentage
width	width of the layout	optional	integer|percentage
border	size of the layout border	optional	integer
space	space between content and border	optional	integer

FlowLayout arranges the inner regions in the same way as words on a page: the first line is filled from left to right, then does the same for the lines below. Table 4 summarizes the possible attributes of the FlowLayout.

Table 4
FlowLayout attributes.

Attribute	Meaning	Usage	Values
border	size of the layout border	optional	integer
space	space between content and border	optional	integer

TimeLayout shows the inner regions in a time sequence that produces a slide show. Table 5 summarizes the possible attributes of the TimeLayout. The *duration* attribute has a float value that represents number of seconds. TimeLayout is used for browsing platforms that support time sequences for presenting media items, e.g., Timed Interactive Multimedia Extensions for HTML (HTML+TIME) [46] and Synchronized Multimedia Integration Language (SMIL) [47].

Table 5
TimeLayout attributes.

Attribute	Meaning	Usage	Values
duration	play time for a sequence element	optional	integer
repeat	number of times to repeat one sequence	optional	"indefinite"|integer

Table 6 summarizes the possible layout-related attributes for a region used inside a BoxLayout, TableLayout, or FlowLayout. These attributes describe how each referenced region has to be arranged in its surrounding layout. For example, the regions embedded in a layout form a sequence for which the order needs to be specified. For this purpose the *order* attribute is used. Note that for the TableLayout, the cell elements are counted from left to right and from top to bottom. The *sort* attribute specifies the sorting criteria for region instances. For example *alpha(Slice.Technique.tname,ascending)* specifies an alphabetical sorting in ascending order based on the name of artistic techniques. Besides the existing sorting functions like *alpha* and *num*, for alphabetical and numerical sorting, one can use its own sorting function (e.g., a multi-sort for data with different facets). If the sort criteria is not provided, the regions will be arranged in the order in which region content (data) is given by the data collection phase.

Table 6
Layout-related region attributes inside BoxLayout/TableLayout.

Attribute	Meaning	Usage	Values
valign	vertical alignment	optional	"left"|"center"|"right"
halign	horizontal alignment	optional	"top"|"center"|"bottom"
ratio	space to be filled	optional	percentage
order	order in the sequence	optional	integer
sort	sorting criteria	optional	string
wml_visible	show on same card	optional	Boolean
wml_description	anchor description	optional	string

Even though most attributes are platform-independent, there are platform-dependent attributes in order to consider the specific card-based structure of WML presentations. The optional attribute *wml_visible* determines whether in a WML presentation a region should be shown on the same card. If not, it is put onto a separate card that is accessible by an automatically generated hyperlink, the text of which is defined in *wml_description*. The *wml_description* attribute can refer to a constant string or one of the simple slices that give some of the content for a region. Note that this kind of content separation provides scalability by fragmenting the presentation according to the small displays of WAP phones.

Table 7 summarizes the possible layout-related attributes for a region used inside a FlowLayout. It is a subset of the previous set of attributes.

Table 7
Layout-related region attributes inside FlowLayout.

Attribute	Meaning	Usage	Values
order	order in the sequence	optional	integer
sort	sorting criteria	optional	string
wml_visible	show on same card	optional	Boolean
wml_description	anchor description	optional	string

Table 8 summarizes the possible attributes for a region used inside a TimeLayout. The *begin*, *duration*, and *end* attributes have float values that represent the number of seconds.

Table 8
Layout-related region attributes inside TimeLayout.

Attribute	Meaning	Usage	Values
begin	(absolute) start time	optional	float
duration	play time	optional	float
end	(absolute) end time	optional	float

Table 9 presents some of the possible style attributes. These attributes refer to the font characteristics (e.g., size, color), background, link colors, etc. The definition of these attributes is inspired from the very elaborate standard to represent style information Cascading Style Sheets (CSS) [48]. We chose not to use directly CSS because not all considered platforms support CSS (e.g., cHTML browser or WML browser), in which case one needs to translate the abstract representation of style to a concrete implementation (e.g., cHTML or WML without CSS).

Table 9
Style attributes.

Attribute	Meaning	Usage	Values
font-family	the family of a font	optional	“times”|“helvetica”| ...
font-style	the style of a font	optional	“normal”|“italic”
font-size	the size of a font	optional	“small”|“medium”|“large”
font-color	the color of a font	optional	“red”|“green”|...
font-weight	the weight of a font	optional	“normal”|“bold”|...
background-color	the color of the background	optional	“red”|“green”|...
link-color	the color of a not-visited link	optional	“red”|“green”|...
visited-color	the color of a visited link	optional	“red”|“green”|...
...			

We chose for an RDF-based representation of the presentation style (instead of using CSS directly), as this allows sharing the RDF data model for the application's models (instead of using the CSS data model), and thus enables reusing the same parsing tools and adaptation techniques (appearance conditions) proposed for the previous models also for the presentation model.

The layout managers need to be instantiated in order to be used in the PM. The layout manager instances are used for complex regions. Also when referencing a region (or set of regions) one needs to define values for the layout-related region attributes corresponding to the layout associated to the container region.

Figure 15 shows an excerpt of the PM for the running example. Regions are depicted as rectangles. There are two top-level regions: *RegionFullT* and *RegionFullA*. *RegionFullT* and *RegionFullA* are owned by *Slice.Technique.main* and *Slice.Artifact.main*, respectively. We use the convention to denote the region (long) name by *Region.<Slice full name>.<Region short name>*. The short name of a region can be omitted from its full name, if the full name unambiguously identifies the region. The full name of *RegionFullT* is *Region.Slice.Technique.main.RegionFullT*. As the full names are quite long in the rest of the explanation it is used the short name of regions when these short names are available.

The region *RegionFullT* aggregates (by means of slice aggregation relationships) three regions: one contains the technique name, one contains the technique description and, one contains the set of pictures that exemplify a painting technique. As simple regions, the first two regions do not need a layout. The third region, a complex region, has a *TableLayout* specified for arranging the set of pictures. All three regions are arranged using a *BoxLayout* specified in the *RegionFullT*. The style information is given by the *DefaultStyle*. As can be seen from the figure the inner regions do not have the style information explicitly defined which means that they inherit the style information from the container region. In a similar manner the region *RegionFullA* is defined. The region navigation relationship connects *RegionBottomA* with *RegionFullA*.

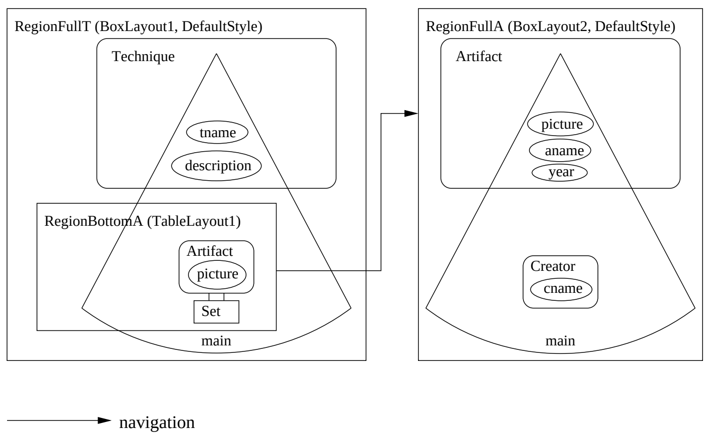

Fig. 15. Presentation model.

Figure 16 shows some of the layout attributes and layout-related region attributes for our running example. *RegionFullT* has a BoxLayout with two attributes defined: *axis* with value *y* which indicates that this layout has a vertical arrangement and *width* with value *100%* which means that this layout will completely fill the width of its container. As *RegionFullT* is a top-level region, the container is the user's display. In *BoxLayout1* there are three regions embedded in the order specified by the *order* attribute. All three regions have the *halign* layout-related attribute defined in order to specify that their horizontal alignment will be centered. The third layout, *RegionBottomA* has two attributes defined: *cols* with value *3* which indicates that this layout has three columns and *width* with value *100%* which means that this layout will completely fill the width of its container. The container is in this case *Region-*

FullT. *RegionBottomA* contains pictures for which the horizontal alignment is centered.

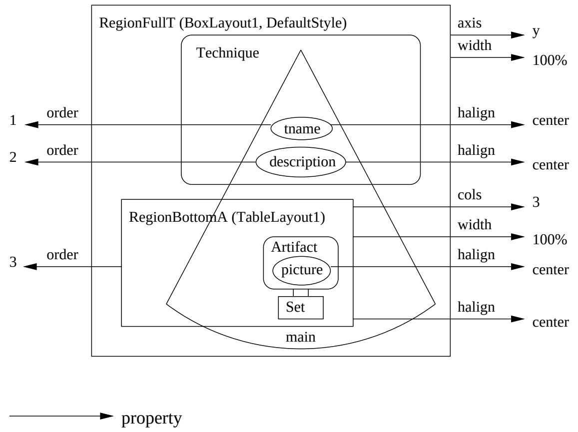

Fig. 16. Layout and layout-related region attributes.

The PM presented in Figure 15 depicting the main regions for techniques and artifacts can be refined to a specific artistic domain. Figure 17 shows the specialization (in a type hierarchy) of the previous PM to the painting domain.

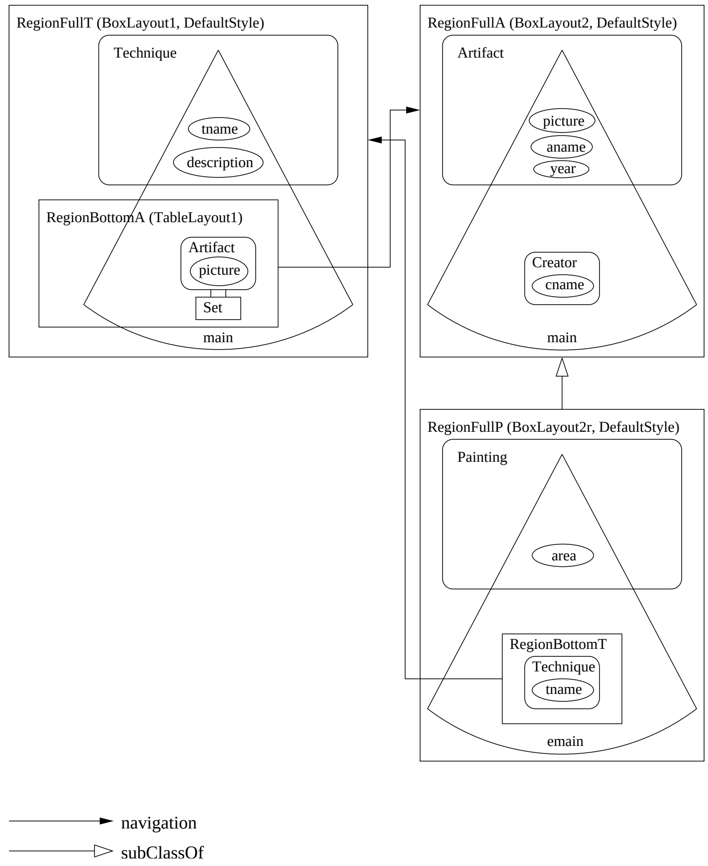

Fig. 17. Specialization in the presentation model.

Regions are specialized by the *subClassOf* property. For example, the region

RegionFullA is specialized by the region *RegionFullP*. *RegionFullP* inherits all the region relationships of *RegionFullA* and adds three new region relationships to it: two region aggregations and one region navigation. The aggregation relationships refer to the regions *Region.Slice.Painting.area* and *RegionBottomT*. As *RegionFullP* contains more regions than *RegionFullA*, the *BoxLayout2* is replaced with *BoxLayout2r* which among other things specifies in which order the added regions are placed. The navigation relationship links backwards the *RegionBottomT* with *RegionFullT*.

PM adaptation selects layouts or styles from PM to be used in the presentation. Figure 18 shows two adaptation examples in PM. In one example, depending on the size of the screen, the *RegionBottomA* uses a *BoxLayout* for PDA and a *TableLayout* for PC. The small screen size of the PDA requires a vertical arrangement of the data. In the other example the *DefaultStyle* uses medium fonts for a user with a average level of vision and large fonts for a user with a low level of vision. Other possible adaptation examples are: increasing the font of links for users with limited manual dexterity, eliminate colors for color-blind users, etc.

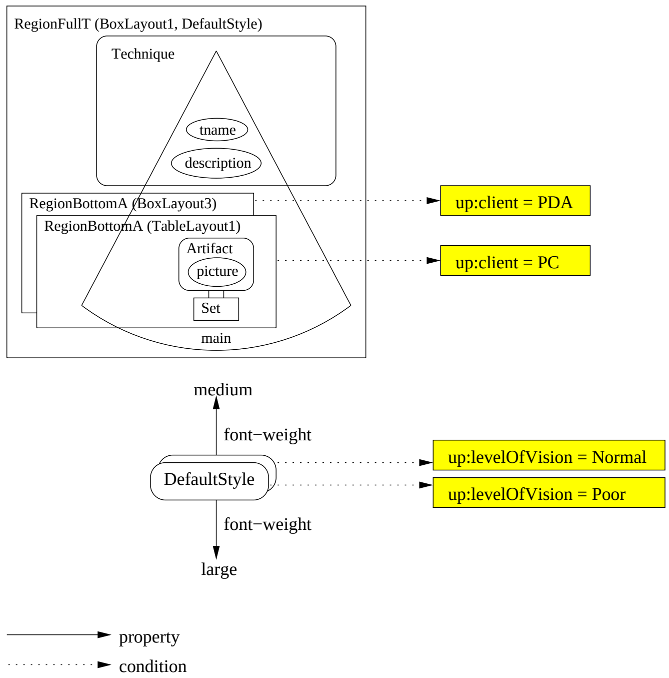

Fig. 18. Adaptation in the presentation model.

4.4 Implementation

The implementation of the static variant of the Hera presentation generation phase is based on several data transformations specified by XSLT [49] stylesheets [50]. These transformations operate on the RDF/XML [51] serialization of the RDF models. The XSLT processor used for interpreting XSLT stylesheets is Saxon [52]. Figure 19 shows the transformation steps for the static variant of the Hera presentation generation phase. Each transformation step has a label associated with it. Some of these transformations have substeps which are labeled using a second digit notation.

In Figure 19 there are two types of dashed arrows: “is used by” to express that an RDFS model is used by another RDFS model and “has instance” to denote that an RDFS model has as instance an RDF model. A model vocabulary, a model, a model instance, and the generated presentations are depicted by rectangles. The transformation specifications are represented by ovals.

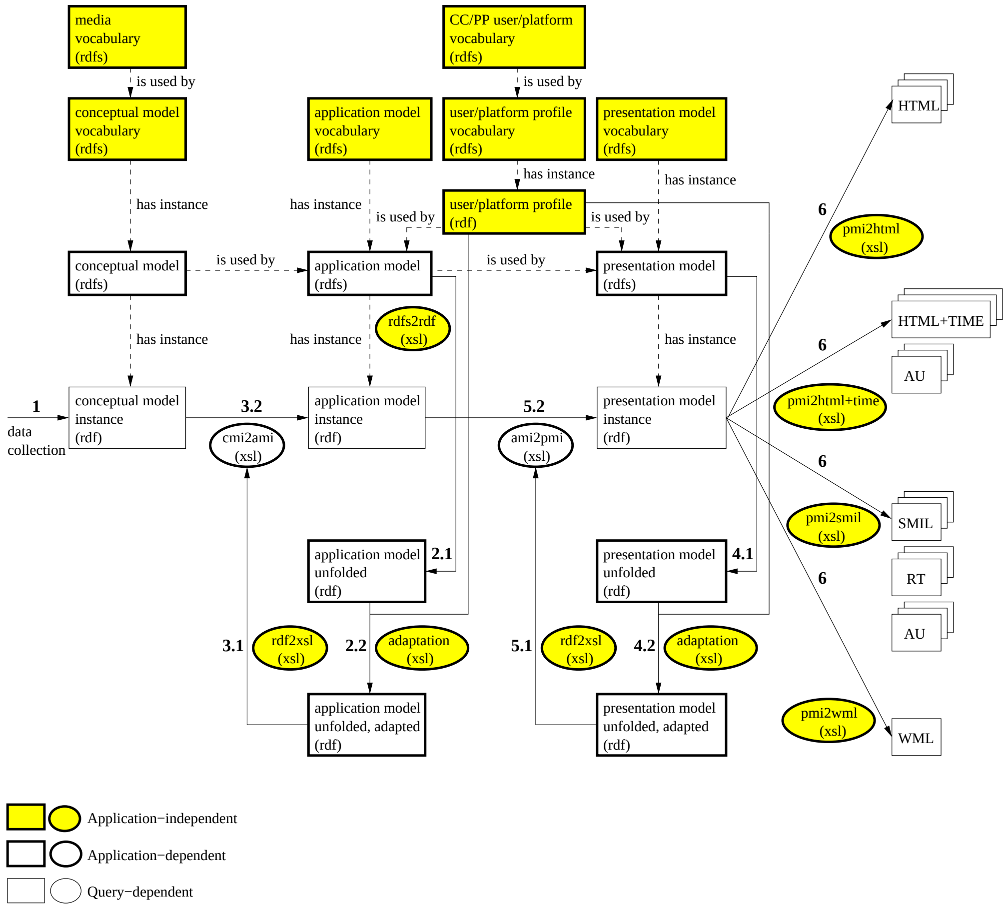

Fig. 19. Presentation generation using XSLT.

There are three types of model/transformation specifications: application-independent, application-dependent, and query-dependent. The application-independent specifications do not refer to SWIS models (CM, AM, and PM), the application-dependent specifications refer to SWIS models, and query-dependent specifications refer to the SWIS models and the retrieved data (e.g., model instances). One can note that the query-dependent transformations are also application-dependent transformations. Transformations that are application-independent are also called generic transformations. Transformations that are application-dependent are also referred as specific transformations.

The input to the presentation generation phase is the conceptual model instance (CMI), i.e., the data retrieved in response to a user query. This data is produced in the *data collection* phase from a given set of input sources. This is step 1 in the figure and is not described here. More information on step 1 can be found in [33]. At the current moment CM and media adaptation are performed in the AM adaptation. Future implementations will separate the CM and media adaptation from the AM adaptation.

Step 2, the *AM generation*, builds an adapted AM template. This step contains two substeps: the *AM unfolding* and the *AM adaptation*.

Step 2.1, the *AM unfolding*, generates the AM template. The AM template represents the structure of an AM instance (RDF) based on the AM schema (RDFS). Such a template will ease the specification of an XSLT stylesheet used to convert a CM instance (CMI) to an AM instance (AMI). By unfolding the AM we mean repeating the process of adding properties inside the subject classes until slice references or media items are reached. In this way one obtains an AM template which will be filled later on with appropriate instances.

Figure 20 presents an excerpt of an XSLT stylesheet. It is a template which iterates over all slices in order to unfold them. One of the difficulties that we encounter when specifying transformations is the fact that it is not easy to keep track of the structure of the resulting document, while traversing the input document. This difficulty can be alleviated if one enforces in the built templates a certain order of parsing the input document.

```
<xsl:template match="rdf:RDF">
  <rdf:RDF xml:lang="en"
           xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
           xmlns:cm="http://wwwis.win.tue.nl/~hera/ns/cm#"
           xmlns:type="http://wwwis.win.tue.nl/~hera/ns/type-system#"
           xmlns:am="http://wwwis.win.tue.nl/~hera/ns/am#">
  <!--
   !! select all slices
   -->
  <xsl:variable name="var_slices"
                select="//rdfs:Class[rdfs:subClassOf/@rdf:resource='.../am#Slice']"/>
    <xsl:for-each select="$var_slices">
      <!--
       !! process recursively all slices
       -->
      <xsl:call-template name="unfoldclass">
        <xsl:with-param name="var_class_name" select="@rdf:ID"/>
      </xsl:call-template>
    </xsl:for-each>
  </rdf:RDF>
</xsl:template>
```

Fig. 20. An excerpt of an XSLT stylesheet.

Step 2.2, the *AM adaptation*, executes the adaptation specifications on the AM template. The transformation stylesheet of this step has two inputs: the AM template and the UP. The UP attributes are replaced in the conditions by their corresponding values. The slices that have the conditions not valid are discarded and the hyperlinks pointing to these slices are disabled.

Step 3, the *AMI generation*, instantiates the AM with the retrieved data. This step is composed of two substeps: the *AMI transformation generation* and the *AMI creation*.

Step 3.1, the *AMI transformation generation*, builds the XSLT stylesheet that will convert a CMI to an AMI. This step uses an XSLT stylesheet that will generate another XSLT stylesheet. One should note that an XSLT stylesheet is a valid XML file that can be produced by another XSLT stylesheet. This technique was also successfully used in the previous version of the implementation which was XML-based [53]. This transformation is based on the *owner* of a slice and the *concept attribute* of a simple slice. The following name convention is used: a slice instance name (e.g., *Slice.Painting.main_ID1*) is obtained from the slice name (e.g., *Slice.Painting.main*) concatenated with the suffix (e.g., *ID1*) of the associated concept instance identifier (e.g., *Painting_ID1*). The implemented algorithm is straightforward: instantiate all slices for all the corresponding retrieved concept instances and each time a slice is referenced add its identifier based on the above name convention.

The transformation used in this phase is a generic one, but the output that it produces is used for a specific transformation (the next step).

Step 3.2, the *AMI creation*, converts the CMI to an AMI. The XSLT stylesheet obtained in the previous substep is applied to the CMI to yield an AMI. As opposed to the previous transformations, this stylesheet will operate for inputs and outputs that are both query-dependent. For each query, Hera will dynamically instantiate the AM with the query result, i.e., a CMI.

The PM-related transformation steps (steps 4 and 5) are realized in a similar manner as the AM-related transformation steps (steps 2 and 3).

Step 4, the *PM generation*, builds a PM template. This step contains two substeps: the *PM unfolding* and the *PM adaptation*.

Step 4.1, the *PM unfolding*, generates the PM template. The PM template represents the structure of a PM instance (RDF) based on the PM schema (RDFS). Such a template will ease the specification of an XSLT stylesheet used to convert an AM instance (AMI) to a PM instance (PMI). By unfolding the PM we mean repeating the process of adding properties inside the subject classes until slice references or media items are reached. In this way, one obtains a PM template which will be filled later on with appropriate instances.

Step 4.2, the *PM adaptation*, executes the adaptation specifications on the PM template. The transformation stylesheet of this step has two inputs: the PM template and the UP. The UP attributes are replaced in the conditions by their corresponding values. The layouts and styles that have the conditions not valid are discarded.

Step 5, the *PMI generation*, instantiates the PM with data from the AMI. This step is composed of two substeps: the *PMI transformation generation* and the *PMI creation*.

Step 5.1, the *PMI transformation generation*, builds the XSLT stylesheet that will convert an AMI to a PMI. As in step 3.1, an XSLT stylesheet that will generate another XSLT stylesheet is used. This transformation is based on the *owner* of a region and the fact that simple regions are associated to simple slices. The following name convention is used: a region instance name (e.g., *Region.Slice.Painting.main.RegionFullA_ID1*) is obtained from the region name (e.g., *Region.Slice.Painting.main.RegionFullA*) concatenated with the suffix (e.g., *ID1*) of the associated slice instance identifier (e.g., *Slice.Painting.main_ID1*). The implemented algorithm is straightforward: instantiate all regions for all the corresponding slice instances and each time a region is referenced add its identifier based on the above name convention.

Step 5.2, the *PMI creation*, converts the AMI to a PMI. The XSLT stylesheet obtained in the previous substep is applied to the AMI to yield a PMI. As opposed to the previous transformations, this stylesheet will operate for inputs and outputs that are both query-dependent.

Step 6, the *presentation data generation*, transforms the PMI into code specific for the user's browser. Note that a set of Web pages is generated at-a-time. Some of supported formats are: HTML, HTML+TIME, WML, and SMIL. For each type of serialization a specific stylesheet is used. The stylesheets used for HTML, HTML+TIME, and SMIL utilize the XSLT 2.0 [49] feature to generate multiple outputs (this feature is not supported in XSLT 1.0 [54]). In order to generate multiple outputs the XSLT 2.0 *result-document()* function was used.

For HTML(+TIME), BoxLayout and TableLayout are implemented using tables. An HTML presentation is composed from the *index.html* document (starting point of the presentation) and a set of HTML pages each corresponding to a top-level region.

The *FlowLayout* is supported by any HTML browser (the content of a table cell is automatically wrapped if it doesn't fit one line). *TimeLayout* is supported only by HTML+TIME and SMIL browsers.

For WML, there is only one layout supported, i.e., the *BoxLayout* with a vertical alignment. Because lists are not available in WML, they are implemented as simple sequences of items without any visual cues. To each top-level region corresponds a WML *card*. A WML presentation is composed from a single WML document, *a deck* that contains a set of cards. The first card is the starting point of the presentation.

For SMIL, there is an explicit part to describe the layout of a document. As tables/flow are not supported in SMIL, one needs always to fully define the layout information for *BoxLayout*, *TableLayout*, and *FlowLayout*. The *TimeLayout* was defined using the *seq* container for regions. Hera regions are implemented as SMIL regions. A SMIL presentation is composed from a main SMIL document (starting point of the presentation), a set of SMIL documents each corresponding to a top-level region, a set of RealText (RT) clips, one per each text media, and a set of audio clips (AU), one per each audio media.

5 Dynamic Presentation Generation

The Hera methodology has been extended in order to accommodate more complex forms of user interaction in addition to simple link-following, e.g., interaction by means of rich forms in which the user can enter data [55]. In this way the user can better personalize the SWIS according to his needs, specially regarding the dynamics within a browsing session. Figure 21 shows the "loop" with which we extended the presentation generation to support this additional dynamics and to allow the user to influence the generation of the Web presentation. Note that in response to a user query only one page is generated at-a-time instead of the full Web presentation as is the case for the static variant of the presentation generation phase. Generating one-page-at-a-time allows the system to consider the user input before generating the next Web page. The request contains the (owner) concept instance identifier and the slice type of the next slice to be generated (i.e., the one corresponding to the next Web page).

In order to illustrate the dynamic version of the presentation generation the running example is extended such that it allows the visitor to buy posters of the paintings in the museum. For simplicity we didn't model explicitly the posters, assuming a one-to-one correspondence with the depicted painting. Also, after buying a certain painting, the user will not be presented with the same painting again.

In addition to the data from CM, AM, and PM, interaction requires a support for creating, storing, and accessing data that emerges while the user interacts with the system. This support is provided by means of the *user session* (US). US is composed of the *navigation data model*, *user/platform model*, *form models*, and *variables*.

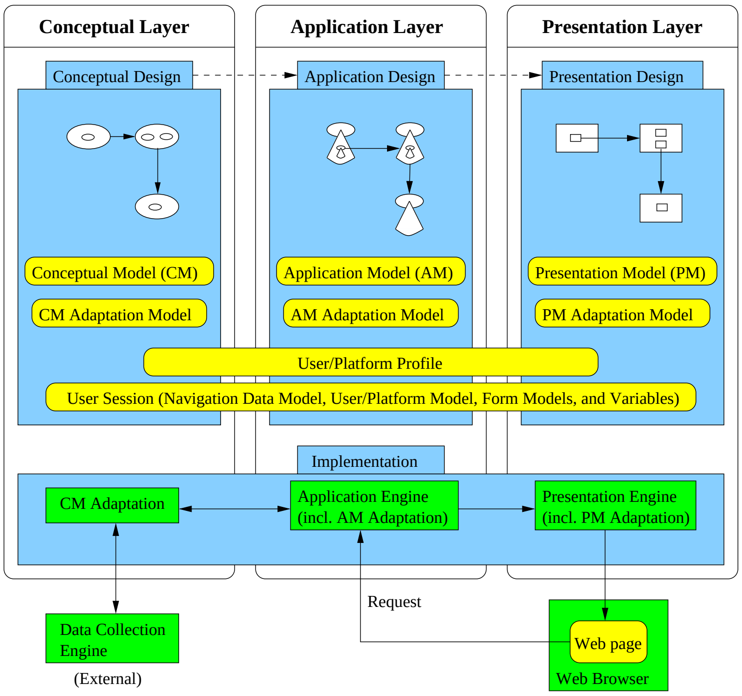

Fig. 21. Presentation generation phase (dynamic).

The purpose of the navigation data model (NDM) is to complement the CM with a number of auxiliary concepts that do not necessarily exist in the CM (although this is the decision of the designer in concrete applications) and which can be used in the AM when defining the behavior of the application and its navigation structure.

The user/platform model (UM) stores user preferences and device capabilities that change during user browsing (e.g., network connection speed, user knowledge on some of the displayed topics, etc.). In Section 4 the UP was defined. The UP-based adaptation is done at the beginning of the user browsing session in order to adapt the CM, AM, and PM. In a similar way the UM is used to adapt the CM, AM, and PM. Differently than for UP, the UM-based adaptation is done before each Web page is generated.

The form models (FM) describe the data that is entered by the user by means of forms. Each form has a so-called form model associated with it. The data input by the user in a form populates the associated form model. Similar to XForms [56], a form separates presentation from content. FM describes the form content. The presentation-related issues of forms are given in the AM.

The session variables are the concept instance identifier, i.e., *instanceid*, and the slice type, i.e., *slicetype*, of the previous slice (the one from which a request originated), and a number of variables to store temporary data created during a user browsing session (e.g., for storing the URIs of newly created resources).

We remark that from the system perspective the concepts in the NDM can be divided into two groups. The first group essentially represents views over the concepts from the CM, the second group corresponds to a locally maintained repository. A concept from the first group can be instantiated only with a subset of instances of a concept existing in the CM, without the possibility to change the actual content of the data. A concept from the second group is populated with instances based on the user's interaction, i.e., the data is created, updated, and potentially deleted on-the-fly. The AM can refer to the concepts from NDM as if they were representing "real" data concepts.

One can note that in the current version of Hera it is not allowed to change the concepts from the CM. The reason is that such updates need to be reflected in the original data sources on which the Hera environment might not have control. Of course in environments in which Hera fully controls the data sources the updates will happen as for the locally maintained repository. Nevertheless, updating the data sources is outside the scope of the current Hera version.

The NDM of our example is depicted in Figure 22. It consists of the following concepts: *SelectedPainting*, *Order*, and *Trolley*[4] (*Painting* is a concept from the CM). The *SelectedPainting* concept is a subclass of the *Painting* concept. It represents those paintings which the user selected in a selection form. We chose to model the selected items as subclasses of their corresponding types (instead of just adding a property to model the selection) because we don't want to change the data structure of the original items. The *Order* concept models a single ordered item consisting of a selected painting (the property *includes*) and the *quantity* represented by an *Integer*. The *Trolley* concept represents a shopping cart containing a set of orders linked by the property *contains*.

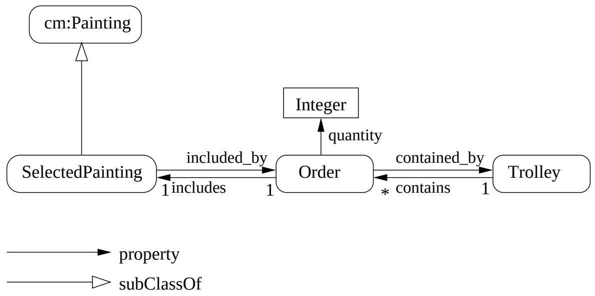

Fig. 22. Navigation data model.

In the example the *SelectedPainting* concept belongs to the group of view concepts whereas both the $Order$ and the $Trolley$ are updatable concepts with the values determined at run-time. This is reflected also in the navigational data model instance (NDMI) depicted in Figure 23 that results from the user's desire to buy 1 poster of the selected painting. The instance *Painting1* comes

[4] Alternative names of the trolley are shopping cart and shopping basket.

from the CM, i.e., it is not (re)created: what is created however, is the *type* property associating it with the *SelectedPainting* concept. Both instances *Order1* and *Trolley1* are created during the user's interaction; they, as well as their properties, are depicted in bold in Figure 23. Note that for presentation purposes (backwards link generation) we also generate for every property its inverse.

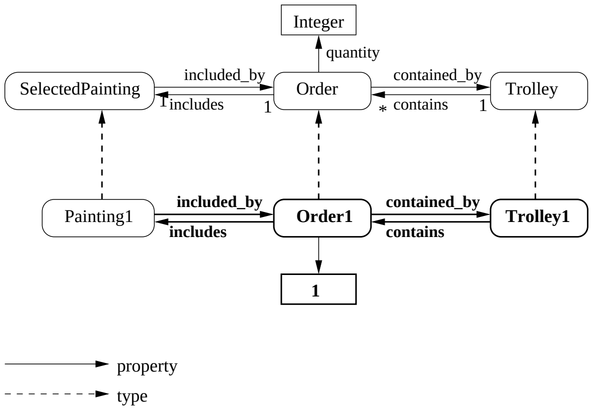

Fig. 23. Navigation data model instance.

The application model vocabulary from Figure 10 was extended in order to support forms. Figure 24 shows these extensions, inspired by the XForms standard. Similar to XForms, a form separates presentation from content. The presentation-related issues of forms are associated to the AM. In AM, a *form* is a particular type of slice which has controls associated with it. The supported form controls (as in XForms) are:

- *Select1* (*S1*), selects one instance from a set;
- *SelectN* (*SN*), selects several instances from a set;
- *Input* (*I*), accepts one line of input text;
- *TextArea* (*TA*), accepts multiple lines as input text;
- *Secret* (*S*), accepts sensitive information entering (e.g., password);
- *Range* (*R*), selects from a sequential display of values (e.g., slide bar);
- *Output* (*O*, displays instance data (e.g., inline text);
- *Upload* (*U*), uploads file or device data (e.g., digital camera images);
- *Trigger* (*T*), activates user triggers (e.g., activate trigger button);
- *Submit* (*SB*), activates data submission (e.g., submit button).

The dynamics of the application is given by a set of AM queries used for selecting, deleting, or updating of data. These queries can be attached to:

- *slices*, to express user-independent updates (e.g., creation of a trolley, as a trolley is created automatically by the system, independent from the user actions);
- *form controls*, to get values for these controls (e.g., select all names of paintings that are not in the trolley);

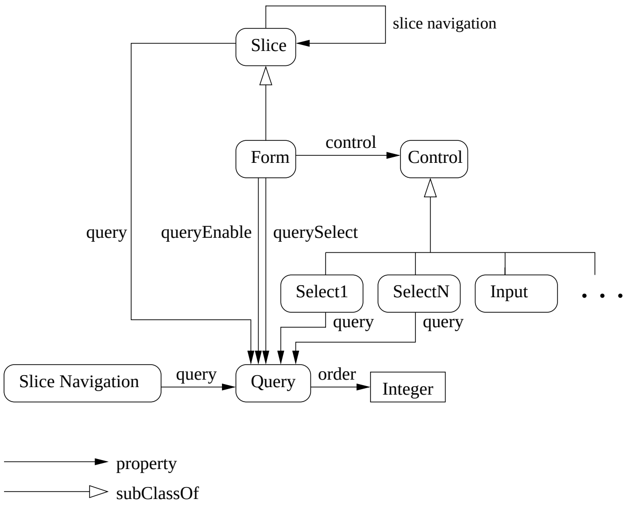

Fig. 24. Extended application model vocabulary.

- *forms*, (1) to enable/disable a form (e.g., if the user has already added all paintings to his trolley, there is no painting left to be offered to the user for the next selection, and therefore the selection form is disabled) or (2) to select the concept instance for the next slice (e.g., after selecting a painting, the main slice of the selected painting is presented);
- *slice navigation*, to express user-dependent updates (e.g., create order and add it to the trolley).

By a query that enables/disables a form it is actually meant a condition that uses some query results for enabling/disabling a form. The identification of the query with the condition is done because the condition usually is a very simple one (in most of the encountered cases it is a comparison of the query result with '0'). An element from AM can have attached a single query or a sequence of queries. The order in which the sequence queries will be executed is given by the *order* attribute.

The content of the form is based on a *form model* (FM), i.e., the schema of the data associated with a certain form. The data of the form that populates (at run-time, based on user actions) the FM is the so-called form model instance (FMI). The mappings (bindings) of the data provided by the form controls to the form model instance are outside the scope of this description as this is done by an external XForms processor. Figure 25 shows an example of a form model and its instance.

Figure 26 shows two form slices that can be embedded in an AM. The short names of the forms are *SelectForm* and *DeleteForm* and the long names are *Slice.Painting.SelectForm* and *Slice.Trolley.DeleteForm*, respectively. The owner of the *SelectForm* is *Painting* and the owner of the *DeleteForm* is *Trolley*. Two queries are used to enable/disable the forms: *QEnableSF* and *QEnableDF*. Both forms have one control field defined *S1* (selects one instance from a set).

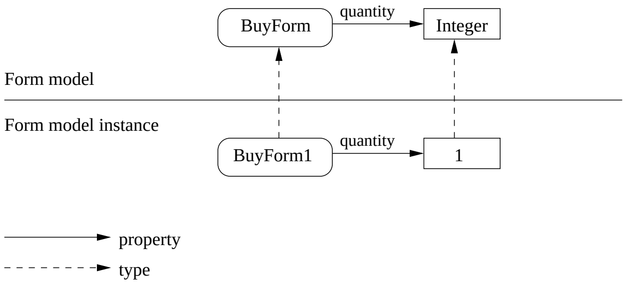

Fig. 25. Form model and form model instance.

The values from which the user makes one selection are given by the queries *QSelectSFPn* and *QSelectDFPn*. The first form has associated *QSelectP* a query that selects a painting instance identifier based on the user's choice. The second form has a slice navigation relationship associated with an update query, i.e., *QDeleteO*.

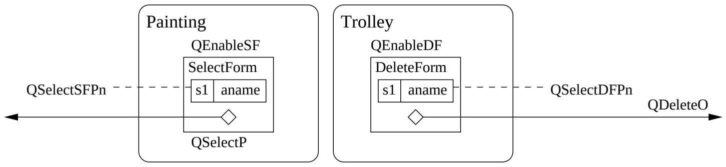

Fig. 26. Form in application model.

Figure 27 shows the application model extended with forms. The main slice of a painting depicts information related to the painting. It also contains the *BuyForm*, a form that allows the user to make an order by specifying the quantity of desired posters for the presented painting. In order not to produce too much visual clutter, we do not show in the figure the concept owner of the form (this is the same as the owner of the destination slice when one navigates from that form). The main slice of the trolley displays the orders contained in the trolley. Note that when the user makes an order, this order is immediately added to the trolley. In addition the main slice of the trolley has two other forms *SelectForm* and *DeleteForm*. *SelectForm* is used to select paintings by their name, paintings which do not have posters in the trolley. *DeleteForm* is used to delete orders from the trolley.

Figure 28 shows the input and output types of a query. Variables are used to pass data from a query result to the input of another query. Note that the designer of the AM needs to use these variables as placeholders for data elements that are not known at design-time.

Because models are represented in RDF(S), the AM queries are described using an RDF query language. As an RDF query language it was chosen SeRQL [57], one of the most expressive RDF query languages that supports not only the selection of RDF data but also the creation of new RDF data. In the future we plan to investigate the usage of SPARQL [58] when this new

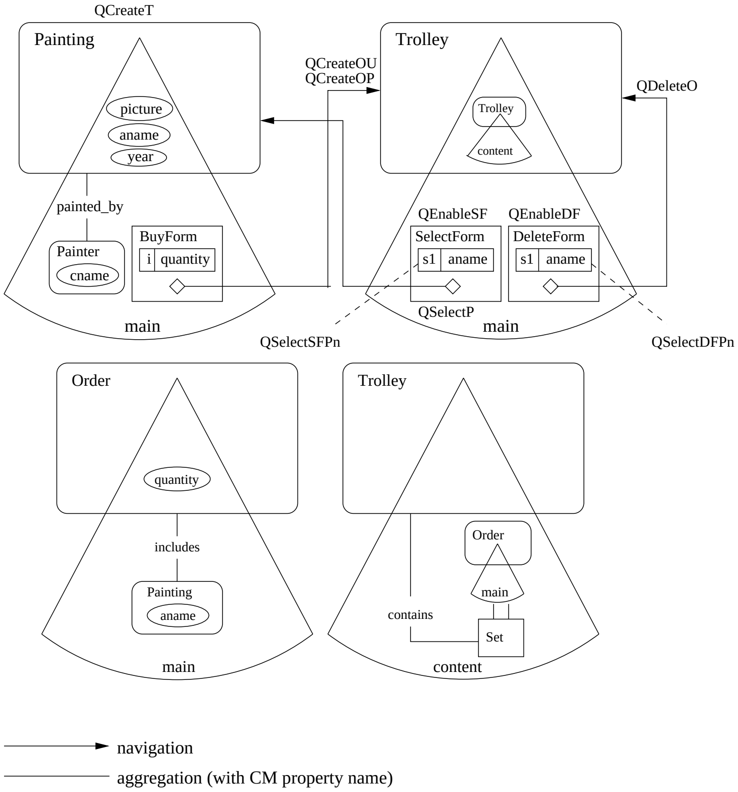

Fig. 27. Extended application model.

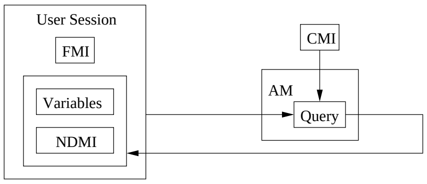

Fig. 28. Query input/output types.

RDF query language proposed by World Wide Web Consortium will reach maturity. In the rest of this section the queries from Figure 27 are presented in their SeRQL syntax. Due to the fact that SeRQL doesn't support nested queries some queries are expressed in RQL [24].

An alternative approach for Se(RQL) queries could have been the use of a Java library for manipulating RDF models (e.g., Jena, Elmo). Using such procedural approach instead of the declarative language like (Se)RQL makes more difficult the maintenance of the implementation. Moreover as there is a lot of effort put nowadays in having a very expressive RDF query language and fast querying engines for it, we would like to benefit from such developments. Nevertheless, in such a case we will limit ourselves to the performance of these existing engines.

Figure 29 shows *QCreateT* a query attached to the main slice of painting.

It is used to create a trolley for a user. The SeRQL was extended with the `new()` function that is able to create a URI unique in the application for a new resource. The new URI is stored in the user session variable *trolleyid*.

```
CONSTRUCT {new()}<rdf:type><ndm:Trolley>
```

Fig. 29. QCreateT (create trolley).

QCreateOU and *QCreateOP* are a sequence of queries attached to the slice navigation from *BuyForm* to the main slice of the trolley. Figure 30 depicts *QCreateOU*, a query that creates a new order. The newly created URI is stored in the user session variable *orderid*.

```
CONSTRUCT {new()}<rdf:type><ndm:Order>
```

Fig. 30. QCreateOU (create order).

Figure 31 shows *QCreateOP*, a query that fills the order properties and adds the order to the trolley. Note that the order is captured in NDM, the owner concept instance identifier of the current slice and the newly generated order identifier are user session variables, and the user input (the poster's quantity) is captured in *BuyForm1*, the form model instance of the form *BuyForm*.

```
CONSTRUCT
     {x}<ndm:contains>{y},
     {y}<ndm:contained_by>{x},
     {y}<ndm:includes>{z},
     {z}<ndm:included_by>{y},
     {y}<ndm:quantity>{v}
FROM
     {session}<var:trolleyid>{x},
     {session}<var:instanceid}{z},
     {session}<var:orderid>{y},
     {BuyForm1}<bf:quantity>{v}
```

Fig. 31. QCreateOP (add order to trolley).

Figure 32 shows *QEnableSF*, a query attached to the *SelectForm* form in order to enable/disable this form. If all paintings have orders associated with them, the *SelectForm* is disabled, as there are no paintings left for user selection. SeRQL was extended with aggregation functions like the *count()* function.

```
(SELECT count(x)
 FROM {x}<rdf:type><cm:Painting>
 WHERE NOT x IN SELECT y
                FROM {session}<var:trolleyid>{v},
                     {v}<ndm:contains>{w},
                     {w}<ndm:includes>{y}) > 0
```

Fig. 32. QEnableSF (condition that enables/disables SelectForm).

Figure 33 shows *QSelectSFPn*, a query attached to the control of the form *SelectForm* in the main slice of trolley. Note that *QSelectSFPn* is a nested query: first the paintings included in the order are computed and the result is subtracted from the set of all the paintings. The query returns the name of the paintings that are not in the trolley.

```
SELECT xname
FROM {x}<rdf:type><cm:Painting>,
     {x}<cm:aname>{xname}
WHERE NOT x IN SELECT y
               FROM {session}<var:trolleyid>{v},
                    {v}<ndm:contains>{w},
                    {w}<ndm:includes>{y}
```

Fig. 33. QSelectSFPn (select paintings (names) that are not in the trolley).

Figure 34 shows *QSelectP*, a query attached to the *SelectForm* in order to select the concept instance that owns the next slice to be presented (i.e., the main slice of a painting). In the future we would like to exploit this selection feature (based on queries) at a more general level, i.e., in the navigation between any two slices and not just between forms (form slices) and slices. In this way the restriction that slice navigation relationships connect slices that have the same owner will be eliminated. Nevertheless one should ensure that only one instance of the destination slice is created.

```
SELECT x
FROM {SelectForm1}<sf:aname>{yname},
     {x}<cm:aname>{yname}
```

Fig. 34. QSelectP (select painting).

Figure 35 shows *QEnableDF*, a query attached to *DeleteForm* in order to enable/disable this form. If the trolley is empty, *DeleteForm* is disabled, as there are no orders to delete.

```
(SELECT count(x)
 FROM {session}<var:trolleyid>{y},
      {y}<ndm:contains>{x}) > 0
```

Fig. 35. QEnableDF (condition that enables/disables DeleteForm).

Figure 36 shows *QSelectDFPn*, a query attached to the control of the form *DeleteFrom* in the main slice of trolley. The query returns the name of the paintings that are in the trolley.

```
SELECT xname
FROM {session}<var:trolleyid>{y},
     {y}<ndm:contains>{x},
     {x}<cm:aname>{xname}
```

Fig. 36. QSelectDFPn (select paintings (names) that are in the trolley).

Figure 37 shows the query *QDeleteO* associated to *DeleteForm* used to delete a selected painting order from trolley. The SeRQL query language was extended with the *DELETE* construct. Basically it is a deletion of statements from an

RDF model. The deletion of resources from an RDF model can be easily done by deleting statements of the form $\{x\}$<*rdf:type*>$\{$*rdf:Resource*$\}$, where `x` is the URI of a resource. A garbage collector will make sure that the properties of the deleted resources will be also removed from the model.

```
DELETE
     {x}<ndm:contains>{y},
     {y}<ndm:contained_by>{x},
     {y}<ndm:includes>{z},
     {z}<ndm:included_by>{y},
     {y}<ndm:quantity>{a}
FROM
     {session}<var:trolleyid>{x},
     {DeleteForm1}<df:aname>{yname},
     {y}<cm:aname>{yname},
     {y}<ndm:includes>{z},
     {y}<ndm:quantity>{a}
```

Fig. 37. QDeleteO (delete selected order from trolley).

In the above queries we did need to extend Se(RQL) with new constructs like URI generators, aggregation functions, and *DELETE* statements. We do hope that future RDF query languages will be equipped with all these constructs.

5.1 Implementation

The implementation of the dynamic variant of the Hera presentation generation phase is based on several data transformations realized in Java. The Se(RQL) queries are executed by Sesame [57] and the data transformations are implemented in Jena [59]. In this way the data transformations exploit more of the RDF(S) semantics given by the Hera models than the ones based on XSLT. A transformation language for XML documents like XSLT cannot use the full RDF semantics stored in the RDF/XML serialization of an RDF model.

Figure 38 shows the transformation steps for the dynamic variant of the Hera presentation generation. Each transformation step has a label associated with it. Some of these transformations have substeps which are labeled using a second digit notation.

In Figure 38 there are two types of dashed arrows: “is used by” to express that an RDFS model is used by another RDFS model and “has instance” to denote that an RDFS model has as instance a certain RDF model. A model vocabulary, a model, a model instance, and the generated presentations are depicted by rectangles. The transformation specifications are represented by ovals. In the same way as for the static variant of the implementation, models and transformation specifications are classified as application-independent, application-dependent, and query-dependent.

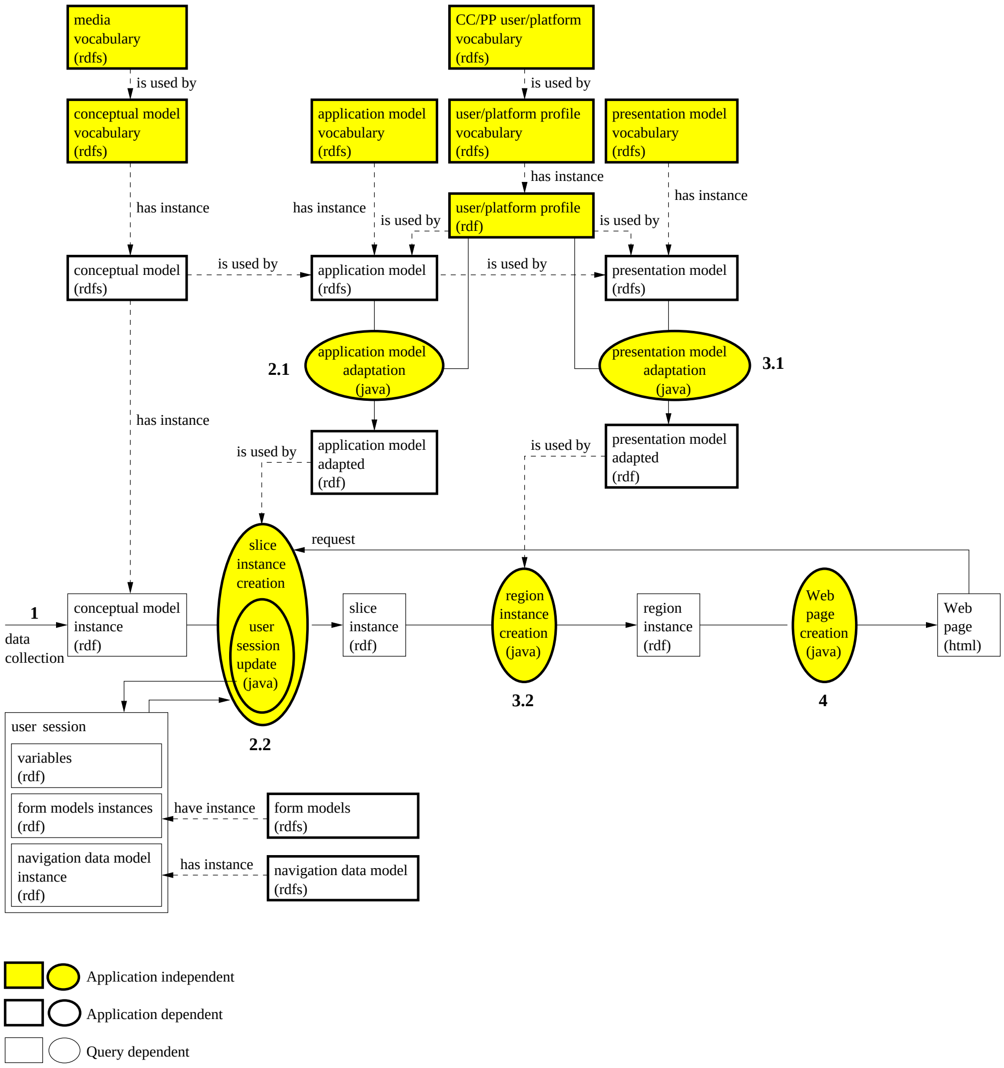

Fig. 38. Presentation generation using Java.

Step 1, the *data collection* phase, is the same as in the static variant of the implementation. The result of this step is the CMI, i.e., the data retrieved in response to a user query. More information on step 1 can be found in [33].

Step 2, the *slice instance generation*, computes a top-level slice instance in response to a user request. This step contains two substeps: the *AM adaptation* and the *slice instance creation*.

Step 2.1, the *AM adaptation*, executes the adaptation specifications on the AM. This transformation has two inputs: the AM and the UP. The UP attributes are replaced in the conditions by their corresponding values. The slices that have the conditions not valid are discarded and the hyperlinks pointing to these slices are disabled. This step is executed only once at the beginning of a user session. In the current version of the implementation, AM adaptation based on the UM is not performed. Future versions of the implementation, that will make use of the user model will execute this step at each user request.

Step 2.2, the *slice instance creation*, creates the next slice instance. The user request provides: the slice type and the concept instance identifier of the slice instance corresponding to the next Web page to be computed, and possibly form model information, in case that request originates from a form. The first user request in a session specifies also the Hera models that will be used in the current session. The queries associated with the slice navigation that initiated the request and the queries associated to the slice to be computed are executed in the *user session update*. Besides updating the NDMI, the *user session update* also stores in the user session the form models and the value of the variables associated to queries.

Step 3, the *region instance generation*, computes the top-level region instance corresponding to the previously computed slice instance. This step contains two substeps: the *PM adaptation* and the *region instance creation*.

Step 3.1, the *PM adaptation*, executes the adaptation specifications on the PM. This transformation has two inputs: the PM and the UP. The UP attributes are replaced in the conditions by their corresponding values. The layouts and styles that have the conditions not valid are discarded. Similar to step 2.1, this step is executed only once at the beginning of a user session. In the current version of the implementation, PM adaptation based on the UM is not performed. Future versions of the implementation, that will make use of the UM, will execute this step at each user request.

Step 3.2, the *region instance creation*, creates the region instance for the previously computed top-level slice instance.

Step 4, the *Web page creation*, transforms the region instance generated in the previous step into code specific to the user's browser. Note that only one Web page is generated at-a-time. At the current moment only HTML is supported by the implementation.

6 Evaluation

The evaluation of the Hera Presentation Generation approach is done at two levels. First, a thorough comparison is made between the static presentation generation and the dynamic presentation generation, the two variants of the Hera presentation generation. After that, we evaluate the Hera presentation generation approach with respect to the requirements set forth for Data-Intensive Interactive Web applications at the beginning of this paper.

The static and dynamic presentation generation in Hera have both their advantages and disadvantages. Figure 10 compares the main features of these two variants of the Hera presentation generation.

Table 10
Static presentation generation vs. dynamic presentation generation.

	Static presentation generation		Dynamic presentation generation
	generation of full Web presentation		generation of one page at-at-time
+	short response time (precomputed pages)	−	longer response time (than for the static presentation generation)
+	deployable on any Web server	−	deployable only on Web servers supporting Java servlets
−	simple user interaction	+	complex user interaction

The static presentation generation builds the whole presentation at once which usually requires a long computation time. Nevertheless if the pages are precomputed and deployed on a Web server the user will not experience any delays during his browsing session. The creation of one-page-at-a-time in the dynamic presentation generation of Hera has as a consequence a larger response time than for the static presentation generation.

Table 11 shows the average response times per page for different applications built with Hera's static presentation generation and Hera's dynamic presentation generation. In order to be able to use Hera's static presentation generation, these applications have as user-system interaction only link following navigation. The experiments have been performed on a DELL Latitude D600 laptop with Intel Pentium M 1600 Mhz processor, and 512 MB RAM. As software we used the Windows XP Service Pack 2 platform with Java 1.4, Apache Tomcat 4 as Web server, Internet Explorer 7 as Web browser, Saxon 5.5.3 for the XSLT transformations, Sesame 1.1-RC2 and Jena 2.0 for the Java transformations.

The pre-computed pages of the static presentation generation can be deployed on any Web server, with or without servlet support. The dynamic presentation generation can be deployed in a Web server that has servlet support, due to the need to consider the user input before each page is generated. Most modern Web servers (like Apache) do support servlets.

Table 11
Average response time per page.

Application	Static presentation generation	Dynamic presentation generation
paintings museum site	34 ms	4065 ms
comic strips site	28 ms	1352 ms
Hera site	25 ms	791 ms

In the dynamic presentation generation we do consider the generation of one page-at-a-time. Due to the granularity offered by slices, in the future we plan to use slices instead of pages as information units exchanged between Web servers and browsers. For the implementation of this extension one can make use of AJAX technology to generate only the part of the page that has changed instead of the whole page, improving thus system performance.

Besides the fact that we can generate truly dynamic content and enable the (dynamic) personalization of the Web application, Hera's dynamic presentation generation can be used for other purposes as well, e.g., i18n (internationalization), i.e., generate content in different languages, accessibility, i.e., allow people with disabilities access the presented information, customized ads (based on the page content generate appropriate advertisements), etc.

In order to improve the performance of the system we did experiment with two methods: building indexes to speed up the query evaluation and data transformations, and precomputation of static information. In the future we would like to experiment with other forms of optimization as preloading (by using heuristics) the content of the next pages to be visited, storing on the client previously computed pages, storing data fragments and performing simple data transformations on the client (e.g., presentation generation), etc.

The static presentation generation has no support for complex forms of user interaction with the system. The only way a user can influence the next page to be displayed is by link-following. In the dynamic presentation generation the user is able to control better the generated presentation by using more advanced forms of interaction like forms. These forms of interaction can be successfully applied in applications like shopping sites or review systems.

The static presentation generation uses XSLT for the data transformations. One of the advantages of XSLT is that it allows to easily update the system by changing only the concerned XSLT stylesheets and not having to recompile the application. Nevertheless XSLT is not able to cope with the full RDF(S) semantics and there is little support to optimize (with respect to the computation time) the data transformations. In order to reduce the computation time for large models we did make use of XSLT keys.

The dynamic presentation generation uses Java for the data transformations. A disadvantage of the dynamic presentation generation is the need to recompile the whole application when a software update is performed. Despite this, the dynamic presentation generation has many advantages like the ability to cope with the full semantics of RDF(S) models and the possibility to optimize the data transformations by defining appropriate data structures and processing units.

It is worth noticing that XSLT and (Se)RQL are declarative languages compared to Java which is a procedural language. In general using the declarative paradigm enables the programmer to focus on what needs to be done and not on how it is done, it is easy to reuse code, and it requires little coding. Nevertheless, such programs are harder to debug and there is little room for optimization (e.g., with respect to program execution time).

Using a procedural language makes the programmer work at a lower level of abstraction than while using a declarative language. In this case the programmer focuses on how to achieve a certain goal and not on what the goal represents. Nevertheless, such an approach allows a full control over the optimization of a program (e.g., with respect to its execution time). Also, procedural programs are relatively easy to be debugged. Disadvantages of procedural languages are that the code is difficult to be reused, and it tends to be rather long.

Several applications were built using the static generation phase of Hera: a portal for a virtual paintings museum, a portal for comic strips, a site for music CDs, etc. The virtual painting museum is based on real-world dataset containing tens of thousands of artifact descriptions made available by the Rijksmuseum in Amsterdam (`http://www.rijksmuseum.nl`), the largest art and history museum in the Netherlands. Due to the fact that we used RDF for our model representations we were able to reuse existing domain models as the ones developed for the museum descriptions in the TOPIA (Topic-based Interaction with Archives) project [35] or previously developed models for specifying device capabilities and user preferences as proposed by the User Agent Profile (UAProf) [36] standard.

Some of the applications built using the dynamic presentation generation phase of Hera are: a shopping site for posters depicting paintings, a review system for the Hera papers, a shopping site for vehicles, etc. For the shopping site for posters we did reuse the models that were developed for the virtual museum portal developed using the static generation phase of Hera which we extended with more complex forms of user interaction. The RDF extensibility feature (e.g., adding new properties to existing resources, specializing previously defined concepts, etc.) enabled us to easily extend the models developed for the static presentation generation.

Some of the most representative applicative requirements for the considered Web applications are: presentation personalization (for user/platform), complex user interaction with the system, and defining the look-and-feel of the application. The managerial requirements relevant for the considered Web applications are scalability, interoperability, reuse, and accessibility. Most of these requirements are seamlessly supported by Hera due to its model-based approach, used adaptation techniques, as well as its reliance on Semantic Web technologies.

The *presentation personalization* is specified by means of appearance conditions in the different Hera models[38]. Using these conditions the designer can adapt the presentation generation so that it is tailored to the user platform and user preferences. These adaptations are specified in the conceptual model, application model and presentation model, selecting only the relevant concepts, appropriate navigation, and layout and style, respectively, that match the user browser and user interests.

Hera allows the development of WIS with advanced forms of *user interaction* with the system by allowing the user to enter information into the system so that he can control the content and navigation structure of the generated presentation[60]. The input is gathered by means of input controls and subsequently stored in the session model. The session model is used at runtime to adapt the conceptual model, application model, and presentation model, as well as input new data into the system in order to make the application more amenable to the current user needs/activities.

The look-and-feel aspects of the application are specified in Hera using *presentation modeling* [31]. The result of this activity is the presentation model, which defines the layout and style information for the generated presentation. The layout information allows building presentations for different platforms: HTML browsers for PC, cHTML browsers for PDA, and WML browsers for WAP phones. Styling the presentation enables the designer to control the finest details of the presentation as, for example, font colors and sizes.

By using Hera, the designer needs to follow a *rigorous methodology* based on a sequence of steps: conceptual design, application design, and presentation design. During these steps models that specify different aspects of the application are being built. In order to ease the specification of a WIS using the Hera methodology we provide a support tool: the Hera Presentation Generator (HPG) [26]. In addition to its design function, this tool also implements the data transformations that *automate* the generation of hypermedia presentations from given specifications and input data.

We measure the scalability of our approach by its ability to handle large amounts of data, and undergo minimum changes when the system requirements change. Model-driven approaches, as Hera, benefit from the fact that an increase in the data volume, does not influence the applications logic, all the data models and associated transformations remaining unchanged. Also, changes to the application logic require modifications in the used models and possibly the data transformations which usually require less time than changing the application code.

Some of the main benefits of using Semantic Web technologies are *reuse* and *interoperability*. Due to the fact that we employed the UAProf standard for modeling the user profile device capabilities and user preferences, we were able to reuse existing profiles made for different mobile devices. Also, the conceptual model is largely based on RDFS data model which allows for the reuse of existing RDFS domain models (as for example the ones retrieved using the Swoogle query engine [61]). All the hera models can be reused while specializing them using the RDFS inheritance mechanism [38].

There are several ways in which a system built using Hera can interact with other systems. First of all the fact that we employ an existing standard for modeling the user profile UAProf, enables us to interoperate with systems that provide such a profile as a service. Also, we can benefit from services provided by external applications, like for example an application service, presentation service, or code generator service [62], given that the meta-models stay the same. For example a code generator service that shares the same presentation meta-model with our application can produce code that is not yet supported by our implementation (e.g., in i-mode format).

The Hera presentation generation supports some aspects related to the accessibility of the built Web application. In this sense Hera aims to make the access to the presented information available for people with disabilities, achieving thus a broader user coverage. For example if the user has a lower level of vision, the application will adapt to the user condition and produce Web pages using large fonts and images. Also, for users with a limited dexterity the font of the links is increased and for color-blinded users colors are removed from the presentation.

The built applications help us validate that Hera provides a scalable approach for developing SWIS, supports presentation personalization based on device capabilities and user preferences, enables advanced forms of user interaction with the system, reuses previously developed design artifacts, specifies and implements in detail the look-and-feel aspects of the application. Moreover, Hera provides a flexible and extensible approach for WIS developing which is seamlessly supported by the use of Semantic Web technologies.

An interesting experiment is to compare the building of data-intensive interactive applications with and without using the Hera methodology. For this purpose, we have experimented with building Web applications using the Hera approach and the traditional, non-model based, approach. For the specification of Hera models we made use of the visual editors of the Hera Presentation Generator (HPG) [26]. As can been noticed from Table 12, at the beginning (first application in the table) it takes more time to build a Web application using the Hera approach, as the developer needs to get familiar with the Hera models. Nevertheless, once the developer gets more experience with building Hera models, he is able to produce the required Web applications faster than in the traditional approach (second application in the table).

Table 12
Average building time.

Application	Method	Average building time
first application: paintings museum site	Hera	7 days
second application: comic strips site	Hera	3 days
paintings museum site	traditional	5 days

Another experiment that we performed is related to the easiness of updating a Web application with new concepts, new links, or new layouts/styles. Based on the data from Table 13 collected for the Web applications that we have implemented, we observe that it takes less time to update the Hera models by adding new concepts, new links, or new layouts/styles, instead of updating the code used in traditional approaches. The average time needed to add a concept is larger than the time needed for adding a link or a layout/style (in both approaches), as one needs to propagate the conceptual changes to the navigational and presentational aspects of the application.

Table 13
Average update time.

Update action	Method	Average update time
add a concept	Hera	1 hour
	traditional	3 hours
add a link	Hera	5 minutes
	traditional	9 minutes
add a layout/style	Hera	12 minutes
	traditional	25 minutes

7 Conclusions

Hera is a model-driven methodology for designing Semantic Web Information Systems. The presentation generation phase of the Hera methodology builds a Web presentation for some given input data. The Hera presentation generation phase has two variants: a static one that computes at once a full Web presentation, and a dynamic one that computes one-page-at-a-time by letting the user influence the next Web page to be presented. The design of both variants uses models that are specified in RDF. The implementation of the static variant is based on XSLT data transformations and the implementation of the dynamic variant is based on Java data transformations.

From the investigated SWIS design methodologies the most mature ones are OntoWebber and (A)SHDM. As OntoWebber and (A)SHDM, Hera is a design methodology which proposes well-defined steps that the SWIS designer needs to follow. Differently than OntoWebber and (A)SHDM, Hera supports more advanced forms of user interaction with the system by allowing the user to input data into the system (in addition to making selections based on existing data) that allows him to control the content of the next page to be generated. Also, Hera supports complex presentation models in which time-based layouts (slide shows) and detailed styling information can be specified.

To our knowledge, none of the examined SWIS design methodologies (XWMF, SEAL, OntoWebber, (A)SHDM) makes explicit the data transformations that need to take place in SWIS built using one of these methodologies. All these methodologies focus on the statical aspects of design, and emphasize less the transformations between models. In Hera we provide a detail description of not only the design models but also the data transformations that use the model specifications in order to build hypermedia presentations.

One of the important contributions of the Hera methodology is the ability to personalize the generated presentation at all application levels: conceptual level, navigational level, and presentation level. In order to support the designer tasks with respect to application personalization, Hera identifies which adaptation aspects can be specified in each of these three application levels. Also, the proposed adaptations use a UAProf user profile that, being based on an industry standard, can be easily reused between applications.

As future work we would like to improve the design and implementation of the Hera presentation generation phase. For the static variant we would like to implement the CM and media adaptation as given in the design specifications as a separate (from AM adaptation) data transformation. The design of the dynamic variant can be extended by adding specifications for UM-based adaptation [63]. With respect to this we anticipate to reuse some of the work done in the adaptive hypermedia field [64]. The implementation of the dynamic variant needs to be extended with other code generators like HTML+TIME, WML, and SMIL.

Also we would like to investigate the use of a declarative RDF transformation language (similar to XSLT but exploiting better than XSLT the RDF semantics). In [65] it is proposed the use of XSLT stylesheets in combination with SeRQL queries (for selections) as a possible RDF transformation language. This hybrid solution is easy to implement and it exploits more of the RDF semantics than XSLT. Nevertheless it relies on the RDF/XML serialization of RDF models and it is less elegant than a solution based on the RDF data model. Lacking an RDF data transformation language based on the RDF data model, we plan investigate the definition and implementation of such a language.

At the current moment Hera doesn't support the requirements phase of the development life cycle of a SWIS. We would like to extend our methodology with a task (activity) model that will specify the activities that can be performed by a user with the system. Once devising a task model one can generate the navigation structure of the application from the task model making easier the design of new application models. The task models can be assigned to a particular user or to a group of users (users that share the same task model) facilitating thus the definition of adaptation at navigation level.

References

[1] T. Isakowitz, M. Bieber, F. Vitali, Web information systems, Communications of the ACM 41 (1) (1998) 78–80.

[2] S. Murugesan, Y. Deshpande, S. Hansen, A. Ginige, Web engineering: A new discipline for development of web-based systems, in: Web Engineering, Vol. 2016 of Lecture Notes in Computer Science, Springer, 2001, pp. 3–13.

[3] P. Barna, F. Frasincar, G. J. Houben, R. Vdovjak, Methodologies for web information system design, in: International Conference on Information Technology: Coding and Computing (ITCC 2003), IEEE Computer Society, 2003, pp. 420–424.

[4] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, M. Matera, Designing Data-Intensive Web Applications, Morgan Kaufmann, 2003.

[5] D. Schwabe, G. Rossi, An object oriented approach to web-based application design, Theory and Practice of Object Systems 4 (4) (1998) 207–225.

[6] A. Diaz, T. Isakowitz, V. Maiorana, G. Gilabert, Extending the capabilities of rmm: Russian dolls and hypertext, in: 30th Hawaii International Conference on

System Sciences (HICSS-30), Vol. 6, IEEE Computer Society, 1997, pp. 177–186.

[7] N. Koch, A. Kraus, R. Hennicker, The authoring process of the uml-based web engineering approach, in: First International Workshop on Web-Oriented Software Technology (IWWOST 2001), 2001.

[8] J. Gomez, C. Cachero, Information Modeling for Internet Applications, Idea Group Publishing, 2003, Ch. OO-H Method: extending UML to model web interfaces, pp. 144–173.

[9] O. Pastor, J. Fons, V. Pelechano, Oows: A method to develop web applications from web-oriented conceptual models, in: International Workshop on Web-Oriented Software Technology (IWWOST 2003), 2003, pp. 65–70.

[10] Y. Jin, S. Xu, S. Decker, Ontowebber: Model-driven ontology-based web site management, in: 1st International Semantic Web Working Symposium (SWWS 2001), Stanford University, 2001, pp. 529–547.

[11] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, Scientific American 284 (5) (2001) 34–43.

[12] R. Klapsing, G. Neumann, Applying the resource description framework to web engineering, in: First International Conference on Electronic Commerce and Web Technologies (EC-Web 2000), Springer, 2000, pp. 229–238.

[13] A. Maedche, S. Staab, R. Studer, Y. Sure, R. Volz, Seal - tying up information integration and web site management by ontologies, IEEE Data Engineering Bulletin 25 (1) (2002) 10–17.

[14] F. Lima, D. Schwabe, Application modeling for the semantic web, in: 1st Latin American Web Congress (LA-WEB 2003), IEEE Computer Society, 2003, pp. 93–102.

[15] P. S. de Assis, D. Schwabe, D. A. Nunes, Ashdm - model-driven adaptation and meta-adaptation, in: Adaptive Hypermedia and Adaptive Web-Based Systems (AH 2006), Vol. 4018 of Lecture Notes in Computer Science, Springer, 2006, pp. 213–222.

[16] T. R. Gruber, A translation approach to portable ontology specifications, Knowledge Acquisition 5 (2) (1993) 199–220.

[17] R. Klapsing, G. Neumann, W. Conen, Semantics in web engineering: Applying the resource description framework, IEEE MultiMedia 8 (2) (2001) 62–68.

[18] O. Lassila, R. R. Swick, Resource description framework (rdf) model and syntax specification, W3C Recommendation 22 February 1999, `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222` (1999).

[19] D. Brickley, R. Guha, Rdf vocabulary description language 1.0: Rdf schema, W3C Recommendation 10 February 2004, `http://www.w3.org/TR/rdf-schema/` (2004).

[20] M. Sintek, S. Decker, Triple - an rdf query, inference, and transformation language, in: First International Semantic Web Conference (ISWC 2002), Vol. 2342 of Lecture Notes in Computer Science, Springer, 2002, pp. 364–378.

[21] A. Maedche, S. Staab, N. Stojanovic, R. Studer, Y. Sure, Semantic portal: The seal approach, in: Spinning the Semantic Web Bringing the World Wide Web to Its Full Potential [outcome of a Dagstuhl seminar], MIT Press, 2003, pp. 317–359.

[22] F. Lima, D. Schwabe, Designing personalized web applications, in: Web Engineering, International Conference (ICWE 2003), Vol. 2722 of Lecture Notes in Computer Science, Springer, 2003, pp. 417–426.

[23] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, L. A. Stein, Owl web ontology language reference, W3C Recommendation 10 February 2004, `http://www.w3.org/TR/owl-ref/` (2004).

[24] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, Rql: a declarative query language for rdf, in: Eleventh International World Wide Web Conference (WWW2002), ACM, 2002, pp. 592–603.

[25] A. R. Hevner, S. T. March, J. Park, S. Ram, Design science in information systems research, Managmenet Information Systems Quarterly (MIS Quarterly) 28 (1) (2004) 75–106.

[26] F. Frasincar, G.-J. Houben, P. Barna, HPG: the Hera presentation generator, Journal of Web Engineering 5 (2) (2006) 175–200.

[27] P. Fraternali, Tools and approaches for developing data-intensive web applications: A survey, ACM Computing Surveys 31 (3) (1999) 227–263.

[28] H. L. Quang, Integration of web data sources: A survey of existing problems, in: Foundations of Databases, Institute of Computer Science, Martin-Luther-University, 2005, pp. 78–82.

[29] O. De Troyer, S. Casteleyn, Modeling complex processes for web applications using wsdm, in: Third International Workshop on Web Oriented Software Technology (IWWOST 2003), 2003, pp. 1–12.

[30] D. Schwabe, G. Rossi, R. Guimaraes, Designing personalized web applications, in: Tenth International World Wide Web Conference, ACM, 2001, pp. 275–284.

[31] Z. Fiala, F. Frasincar, M. Hinz, G. J. Houben, P. Barna, K. Meissner, Engineering the presentation layer of adaptable web information systems, in: Web Engineering - 4th International Conference on (ICWE 2004), Vol. 3140 of Lecture Notes in Computer Science, Springer, 2004, pp. 459–472.

[32] H. Knublauch, D. Oberle, P. Tetlow, E. Wallace, A semantic web primer for object-oriented software developers, W3C Working Group Note 9 March 2006, `http://www.w3.org/TR/sw-oosd-primer/` (2006).

[33] R. Vdovjak, F. Frasincar, G. J. Houben, P. Barna, Engineering semantic web information systems in hera, Journal of Web Engineering 2 (1-2) (2003) 3–26.

[34] D. Connolly, F. van Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, L. A. Stein, Daml+oil (march 2001) reference description, W3C Note 18 December 2001, `http://www.w3.org/TR/daml+oil-reference` (2001).

[35] L. Rutledge, M. Alberink, R. Brussee, S. Pokraev, W. van Dieten, M. Veenstra, Finding the story: Broader applicability of semantics and discourse for hypermedia generation, in: 14th ACM Conference on Hypertext and Hypermedia (Hypertext 2003), ACM, 2003, pp. 67–76.

[36] Wireless Application Protocol Forum, Ltd., Wireless application group: User agent profile, 20 October 2001 (2001).

[37] G. Klyne, F. Reynolds, C. Woodrow, O. Hidetaka, J. Hjelm, M. H. Butler, L. Tran, Composite capability/preference profiles (cc/pp): Structure and vocabularies 1.0, W3C Recommendation 15 January 2004, `http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/` (2004).

[38] F. Frasincar, P. Barna, G. J. Houben, Z. Fiala, Adaptation and reuse in designing web information systems, in: International Conference on Information Technology: Coding and Computing (ITCC 2004), IEEE Computer Society, 2004, pp. 387–291.

[39] Z. Fiala, M. Hinz, K. Meissner, F. Wehner, A component-based approach for adaptive, dynamic web documents, Journal of Web Engineering 2 (1-2) (2003) 58–73.

[40] J. M. Martinez, Mpeg-7 overview, Version 9, ISO/IEC JTC1/SC29/WG11/N5525 March 2003 (2003).

[41] G. Rossi, D. Schwabe, F. Lyardet, Web application models are more than conceptual models, in: International Workshop on the World-Wide Web and Conceptual Modeling (WWWCM 1999), ER 1999, Vol. 1727 of Lecture Notes in Computer Science, Springer, 1999, pp. 239–253.

[42] F. Frasincar, G. J. Houben, R. Vdovjak, An rmm-based methodology for hypermedia presentation design, in: Advances in Databases and Information Systems (ADBIS 2001), Vol. 2151 of Lecture Notes in Computer Science, Springer, 2001, pp. 323–337.

[43] F. Frasincar, G. J. Houben, Hypermedia presentation adaptation on the semantic web, in: Adaptive Hypermedia and Adaptive Web-Based Systems (AH 2002), Vol. 2347 of Lecture Notes in Computer Science, Springer, 2002, pp. 133–142.

[44] P. Brusilovsky, Adaptive hypermedia, User Modeling and User-Adapted Interaction 11 (1-2) (2001) 87–110.

[45] N. Souchon, J. Vanderdonckt, A review of xml-compliant user interface description languages, in: International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS 2003), Vol. 2844 of Lecture Notes in Computer Science, Springer, 2003, pp. 377–391.

[46] P. Schmitz, J. Yu, P. Santangeli, Timed interactive multimedia extensions for html (html+time)-extending smil into the web browser, W3C Note 18 September 1998, http://www.w3.org/TR/NOTE-HTMLplusTIME (1998).

[47] J. Ayars, D. Bulterman, A. Cohen, K. Day, E. Hodge, P. Hoschka, E. Hyche, M. Jourdan, M. Kim, K. Kubota, R. Lanphier, N. Layaida, T. Michel, D. Newman, J. van Ossenbruggen, L. Rutledge, B. Saccocio, P. Schmitz, W. ten Kate, Synchronized multimedia integration language (smil 2.0) - [second edition], W3C Recommendation 07 January 2005, http://www.w3.org/TR/SMIL/ (2005).

[48] B. Bos, T. Celik, I. Hickson, H. W. Lie, Cascading style sheets, level 2 revision 1 css 2.1 specification, W3C Working Draft 13 June 2005, http://www.w3.org/TR/CSS21/ (2005).

[49] M. Kay, Xsl transformations (xslt) version 2.0, W3C Recommendation 23 January 2007, http://www.w3.org/TR/xslt20/ (2007).

[50] F. Frasincar, G. J. Houben, P. Barna, C. Pau, Rdf/xml-based automatic generation of adaptable hypermedia presentations, in: International Conference on Information Technology: Coding and Computing (ITCC 2003), IEEE Computer Society, 2003, pp. 410–414.

[51] D. Beckett, Rdf/xml syntax specification (revised), W3C Recommendation 10 February 2004, http://www.w3.org/TR/rdf-syntax-grammar/ (2004).

[52] M. Kay, Saxon (the xslt and xquery processor), http://saxon.sourceforge.net (2006).

[53] F. Frasincar, G. J. Houben, Xml-based automatic web presentation generation, in: WebNet 2001 World Conference on the WWW and Internet (WebNet 2001), AACE, 2001, pp. 372–377.

[54] J. Clark, Xsl transformations (xslt) version 1.0, W3C Recommendation 16 November 1999, http://www.w3.org/TR/xslt (1999).

[55] G. J. Houben, F. Frasincar, P. Barna, R. Vdovjak, Modeling user input and hypermedia dynamics in hera, in: Web Engineering - 4th International Conference (ICWE 2004), Vol. 3140 of Lecture Notes in Computer Science, Springer, 2004, pp. 60–73.

[56] M. Dubinko, L. L. Klotz, R. Merrick, T. V. Raman, Xforms 1.0, W3C Recommendation 14 October 2003, http://www.w3.org/TR/xforms/ (2003).

[57] Aduna, BV, openrdf.org ... home of sesame, http://www.openrdf.org/ (2006).

[58] E. Prud'hommeaux, A. Seaborne, Sparql query language for rdf, W3C Working Draft 20 February 2006 (2006).

[59] Hewlett-Packard Development Company, LP, Jena - a semantic web framework for java, http://jena.sourceforge.net/ (2006).

[60] G. J. Houben, P. Barna, F. Frasincar, R. Vdovjak, Hera: Development of semantic web information systems, in: Web Engineering - 3th International Conference (ICWE 2003), Vol. 2722 of Lecture Notes in Computer Science, Springer, 2003, pp. 529–538.

[61] L. Ding, T. W. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, J. Sachs, Swoogle: a search and metadata engine for the semantic web, in: International Conference on Information and Knowledge Management (CIKM 2004), ACM, 2004, pp. 652–659.

[62] F. Frasincar, P. Barna, G. J. Houben, A Web Service-Oriented Architecture for Implementing Web Information Systems, Frontiers in Artificial Intelligence and Applications, IOS Press, 2006, pp. 219–232.

[63] P. Barna, G. J. Houben, F. Frasincar, Specification of adaptive behavior using a general-purpose design methodology for dynamic web applications, in: Third International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH 2004), Vol. 3137 of Lecture Notes in Computer Science, Springer, 2004, pp. 283–286.

[64] P. De Bra, G. J. Houben, H. Wu, Aham: A dexter-based reference model for adaptive hypermedia, in: 10th ACM conference on Hypertext and Hypermedia (Hypertext 1999), ACM, 1999, pp. 147–156.

[65] J. van Ossenbruggen, L. Hardman, L. Rutledge, Combining rdf semantics with xml document transformations, International Journal of Web Engineering and Technology 2 (2/3) (2005) 248–263.