Semantical Descriptions of Models for Web Design

Peter Barna, Geert-Jan Houben, Flavius Frasincar, and Richard Vdovjak
Technische Universiteit Eindhoven
PO Box 513, NL-5600 MB Eindhoven, The Netherlands
{*pbarna, houben, flaviusf, richardv*}@*win.tue.nl*

Abstract

The use of semantic web languages brings a number of advantages for web engineering methodologies. In this paper we present how in the Hera methodology the design models benefit from description in semantic web languages. We describe how the important models are expressed and how the corresponding meta-models are defined. This is illustrated for the Application Model that specifies the hypermedia navigation over the content in the application, and do so for both the static and dynamic cases. With the support of the dynamics, particularly the interaction of the user with the generated hypermedia presentation, we extend the possibilities to specify the design of a new generation of web applications.

1. Introduction

The current requirements for web applications do ask for a rigorous design and engineering approach. Several design methodologies for web applications have been proposed in the research literature. We mention here OOHDM [10], OO-H [6], UWE [7], WebML [5], and Hera [11]. Characteristic for many of these approaches is their model-based nature. They choose to offer an approach that supports a model-driven design. Typically, one sees models that describe the content (domain), that describe the navigation over that content (hypermedia), and that describe the presentation of that content and its navigation (layout, timing, etc.). In order to get a better grip on the web application design, this model-driven approach proves effective. Specifically, in those data-intensive applications that are constructed from content that is retrieved from a diverse and distributed set of repositories: in that case it is obvious that a schema-based approach is required and the earlier principles from manually constructing a web application are not valid anymore. Most of these design approaches specially target this second generation of web applications.

With the further development of the applications from this second generation, the aspect of dynamics starts playing a more important role. In the second generation the target was much more the presentation of the content in a way that conveyed the appropriate semantics to the application user. In the third generation there is also more attention for the interaction of the user with the presentation: the actions from the user do influence the behavior of the application, and the design models need to reflect that aspect of the communication as well. So, in the next generation of the web design methodologies we see a stronger focus on the design of dynamic web applications.

A second trend that we observe in the design of web applications is the use of semantic web technologies. As we already recognized quite early in the process of the development of our Hera methodology, some of the languages and techniques that emerged from the semantic web initiative could be extremely useful in the specification of the different aspects of the design models. Specially when it comes to sharing and exchanging information (and their schemas), languages like RDF [8], RDFS [4], and OWL [9] appear to be quite effective.

In this paper we address the issue of the semantical description of the design models. We explain and illustrate how the semantical descriptions can play a prominent role in the specification of the web application. Furthermore, we show how the (meta-)models for these models can be described and benefit from the use of RDF(S) and OWL. We do so in terms of our Hera methodology and for this paper we particularly focus on the application model, i.e. the model that specifies the (hypermedia) navigation over the content. In Section 2 we shortly recap the essentials of Hera, while Section 3 covers the models of the semantical layer (the content models). Then, in Section 4 we take a closer look at the application layer with the models for the navigation, first for the static case, then for the dynamic case. Section 5 presents the implementation of the tools that support the methodology, before Section 6 concludes our view on the role of semantical descriptions in web design.

2. Hera

Hera [11] is a model-driven methodology that defines a set of design steps that need to be taken to build a web application. In the design steps a set of models is constructed describing certain aspects of the designed web application. The models are used within the specification of an automatic data transformation process starting with data retrieval and producing hypermedia presentations in an end-user format (for instance HTML).

The Hera methodology recognizes the following phases and models:

- Conceptual (semantic) design. The main output of this phase is the Conceptual Model (CM) describing the data structure of the overall data content used in the designed web application. The Media Model associates data items from CM with media types from the Media Vocabulary.

- Application design. In this phase the designer creates the Application Model (AM) defining the navigation structure and behavior of the application.

- Presentation design. In this phase the designer specifies the layout and rendering of presented content materialized in the Presentation Model (PM). It is out of scope for this paper.

The process of data transformations starts with posing a query to the data repository. The process of presentation generation (see Figure 1) has the following steps:

- Data retrieval and CM instance generation, where the required data is collected from data sources and transformed into an instance of CM (CMI). This is performed within the Semantic Layer (CM defines the semantics of the data content).

- AM instance generation, where the data is transformed into an AM instance (AMI) using the AM. AMI is generated within the Application Layer, because the AM defines the application structure (navigation and behavior).

- Presentation generation, where the AMI is transformed using the PM into a presentation in a concrete format. Presentations are generated within the Presentation Layer.

Meta-models are used to define the basic modelling primitives that are used in all concrete models specifying the designed applications. All models and meta-models are expressed in RDFS [4]. In the next sections we describe meta-models and models for the Semantic Layer and the Application Layer.

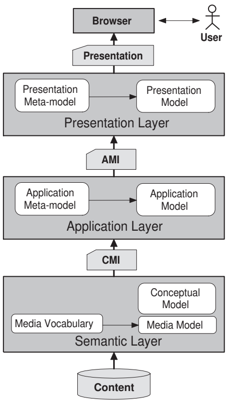

Figure 1. Data transformations using models in Hera

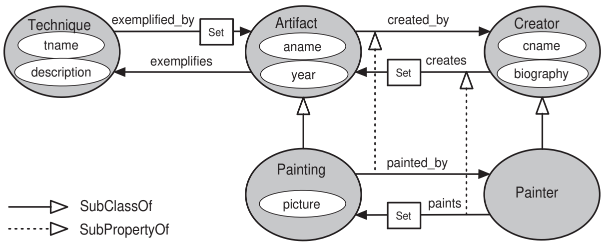

Figure 2. An example of CM using Hera's graphical notation

3. Models for the Semantic Layer

CM describing the semantics of the data content, is the core model of the Semantic Layer. Next to CM, the Media Model (MM) associates displayable data attributes with appropriate media types.

Both CM and MM are expressed in terms of RDFS, by means of concepts, their attributes (literals), and properties (including predefined properties as `rdfs:subClassOf`, `rdfs:subPropertyOf`, and `rdf:type`). Hence, RDFS is a meta-model for CM and MM. In addition, the Media Vocabulary defines the hierarchy of media types used in MM.

For CM we use Hera's own concise graphical notation that mimics RDFS, where darker ovals represent RDF(S) concepts, white ovals inside concepts represent literal attributes, and arrows represent concept properties. The `Set` construct is a shortcut for the $0...*$ cardinality that is added to the RDFS meta-model. An example of CM in this notation is given in Figure 2. The example describes a simple domain of artifacts, authors, and used techniques with specializations for painters and paintings.

Figure 3 shows a MM defining media types for the exam-

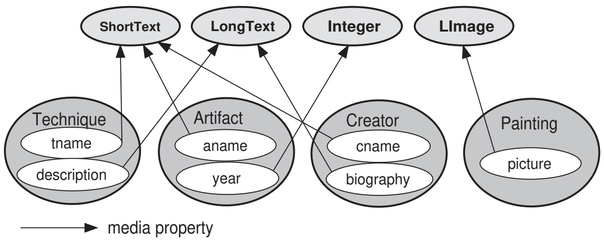

Figure 3. An example of the Media Model

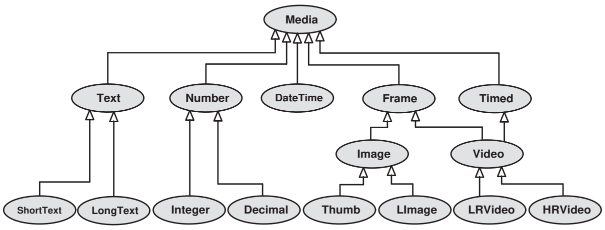

Figure 4. Media Vocabulary

ple CM. It contains associations between literal attributes from CM and media types from Media Vocabulary shown in Figure 4.

4. Models for the Application Layer

The core model of the Application Layer is the AM specifying the (dynamic) navigation view over the conceptual data. We first explain the basic (static) AM using the example from the previous section and the Application Metamodel. Then we explain how navigation dynamics is specified within AM.

4.1. Static AM

Slices are the basic building blocks of AM. They specify the structure of navigation nodes (e.g. pages). Every slice is based on a single concept from CM (owner concept) and represents a meaningful collection of information given by selected attributes of the owner concept, or attributes of related concepts (connected by a property(ies) in CM). A slice is a composite structure that represents a whole page or its fragment. Names of slices are usually derived from their owner concept and their purpose.

Navigation edges (e.g. links) are modelled as slice references that have anchors associated with some slice components. Reference targets are (top-level) slices. A slice can contain components of the following types:

- *attribute* of the owner concept that displays the data value of an attribute;
- *foreign attribute* (attribute of a related concept) that displays the data value of an attribute of a related concept instance given by a CM property;
- *foreign slice* based on a related concept that displays its own content;
- *link anchor* based on foreign attribute, or foreign slice, and
- *sets* of links, foreign attributes, or foreign slices.

For AM Hera uses its own graphical language (inspired by RMM [2] notation) that facilitates the design process. However, the AM specifications are in Hera tools translated into RDFS.

4.1.1 Example of Static AM

An example AM is given in Figure 5. All mentioned types of slice components appear there. The initial slice `Technique.Main` is marked by the black triangle. Ovals represent concepts from CM, where large ovals are owner concepts of slices. Small ovals within slices are related concepts.

The `Technique.Main` slice displays the values of the `tname` and `description` attributes of the instance of `Technique` (for this simple example we assume that there is only one instance of the `Technique` concept in the data content). Furthermore, the set of painting names (`Painting.aname`) is displayed. The selection of proper instances is determined by the `exemplified_by` property. A set of links is associated with `Painting.aname` as link anchoring element, and `Painting.Main` as a target slice. Hence, every instance of the painting (`Painting.aname`) is associated with a link instance navigating to a concrete `Painting.Main` instance displaying more information about the concrete painting.

4.1.2 Meta-model for Static AM

Figure 6 shows a meta-model for AM. The meta-model is presented using UML notation. Although it is currently in RDFS, its extension to OWL, allowing to express the constraint(s) shown in the figure, is under development. The property concepts are in the figure distinguished from other concepts by darker color.

All slices defined in a concrete AM (for instance top-level `Technique.Main` or `Painter.Info`) are subclasses of the general `Slice` concept. Moreover, all slice components are modelled as slices (e.g. attributes like `Technique.tname` or foreign attributes and their sets like `Painting.aname`, etc.) as well.

The `slice-ref` property is the basic means of slice nesting. All sub-components of a slice are associated to the

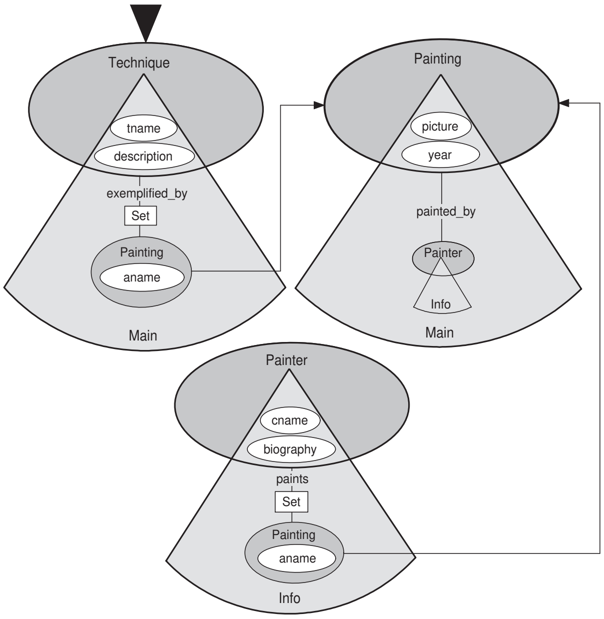

Figure 5. An example of static AM expressed in Hera's graphical language

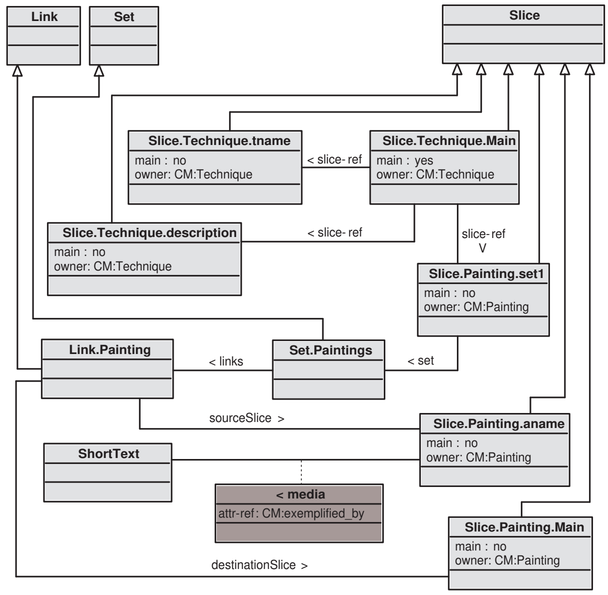

Figure 7. The Technique.Main slice specification

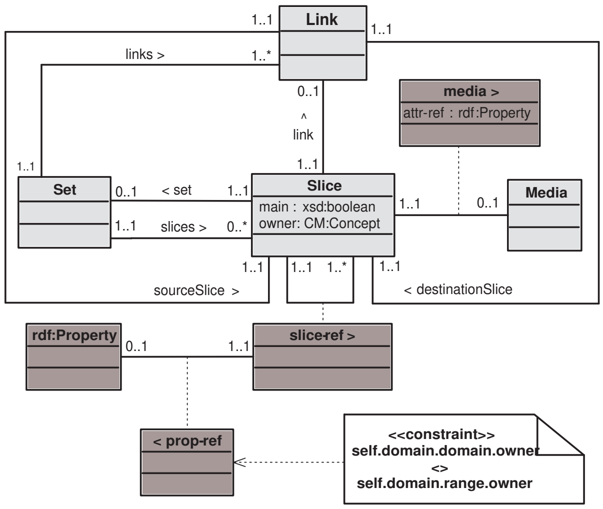

Figure 6. Application meta-model in RDFS and OWL using the UML notation

slice as another slices by means of properties that are subproperties of `slice-ref`. Sub-slices can represent links (if it has the `link` property), attributes (if it has the `media` property), nested slice, or sets of links or slices (if it has the `set` property). The `attr-ref` property of `media` determines an attribute from CM, and (sub-class of) the `Media` concept determines the media type of the attribute. The `prop-ref` property gives a CM property connecting the owner concept of the parent slice with the owner concept of the nested slice. The constraint attached to it limits the existence of `prop-ref` only for such slices related by `slice-ref` that have different owner concepts.

Figure 7 shows the `Techique.Main` slice specification implemented as a specialization of AM meta-concepts. The `Techique.Main` slice and its components are subclasses of `Slice`. The components are of types attributes (e.g. `Slice.Technique.tname`, its `media` property is not shown due to the lack of space) and set (`Slice.Painting.Set1` that has a `set` property `Set.Paintings`). The set of paintings is constructed from the link set (`Link.Painting`) that is based on the set of painting names (`Slice.Painting.aname` with its `media` property defining its media type and the property `exemplified_by` connecting the `Technique` and `Painting` concepts).

4.2. Dynamic AM

In this section we explain how the specification method for the Application Layer, concretely the modelling primitives of AM specified in its meta-model, is extended to allow the design of dynamic web applications. We define the dynamics as the ability of the navigation structure to change depending on the user's actions and the data (s)he enters. An example of such a dynamic pattern is a temporarily stored selection of items, e.g. realized in the form of a virtual shopping basket. Another example is a dynamic personalized navigation view based on a user model (as we typically see in applications that offer user adaptation e.g. [3]).

We need to introduce primitives for modelling input forms, and for operations that will process entered data and based on this change the state of the system (e.g. data content). The notion of application model is thus extended in three ways:

- *Input forms* are added to the set of possible slice components.
- Data content manipulation *operations* associated with input forms are added to the AM specification. They are executed when the forms are filled and submitted by the user.
- *Appearance conditions* can be associated with slices (and its components) to facilitate personalization.

In Hera we have introduced inputs of three basic types that are encapsulated in forms: `select1` allows users to select one value from a list, `selectN` allows user to select a set of values from a list, and `iText` supports textual input. The initial data of these inputs (lists or default text values) can be taken out of the data content or can be constants specified in AM.

Data content manipulation operations are specified in a form of data manipulation queries (in SeRQL [1] in our current implementation) that can use values entered by users in the input forms associated with the operations. They can temporarily or persistently change the data content.

Appearance conditions are attached to slices or slice components: in the case of sets they limit the selection of instances, whereas in the case of single attributes or subslices they determine their visibility. During the generation of application pages the appearance conditions are automatically transformed into data retrieval queries.

4.2.1 Example of Dynamic AM

For the purpose of demonstrating the described modelling primitives we redefine our earlier example now. The application will now initially offer a list of paintings painted by a certain technique (as in the former case), but now it will allow the user to select a number of them. After acknowledging the selection by the user the application will display information about painters, of which paintings appeared in the selection more than once. Although this example is rather hypothetical, it allows us to use all mentioned modelling primitives.

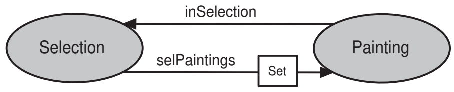

Figure 8. An extension of CM for the support of dynamic navigation

First, we extend the specification of the data as given by CM with auxiliary concepts and properties that will support the navigation dynamics. The instances of the `Selection` concept, and the `selPaintings` and `inSelection` properties will store the user selection, see Figure 8. During the initialization (beginning of session) one instance of `Selection` will be created. Multiple instances of `selPaintings` and its inverse property `inSelection` will specify the user selection of paintings itself.

The AM is in Figure 9. The `Technique.Main` slice contains the `PaintingsForm` input of the `SelectN` type that displays the list of all available paintings painted using the technique. The `initialize` operation called at the beginning of the session creates one instance of `Selection`. The `updateSelection` operation creates instances of `selPaintings` and `inSelection` connecting selected `Painting` instances with the `Selection` instance, and deletes other `selPaintings` and `inSelection` instances storing possible previous selections. The `Selection.Painters` slice uses the `selPaintings.painted_by` joint property to select painters that painted selected paintings. The list of `Painter.Info` instances is limited by the appearance condition to painters, of which paintings appeared in the selection more than once. The `Painter.Info` slice remains the same as in the previous example, so it is not shown here again.

4.2.2 Extension of Meta-model for Dynamic AM

Figure 10 shows the extension of the application meta-model containing additional modelling primitives that allow the specification of dynamic web applications.

A slice can have an associated appearance condition (the `Condition` concept) that is a specialization of `Query`. Specification of the user input elements is provided by the `IForm Input`, `select1`, `selectN`, and `iText` concepts. The input processing is modelled by the (sequence of) `Operation` concept that is a specialization of a (data

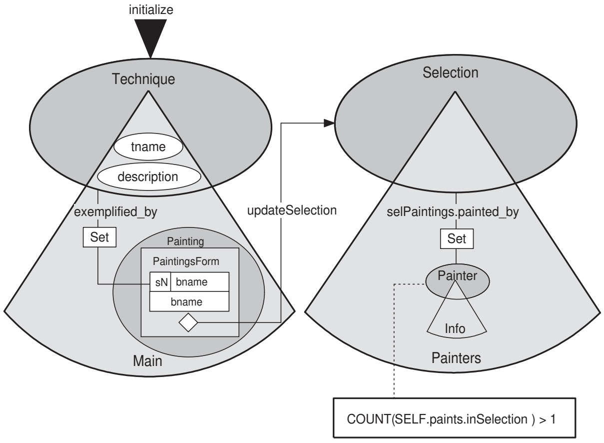

Figure 9. An example of dynamic AM expressed in Hera's graphical language

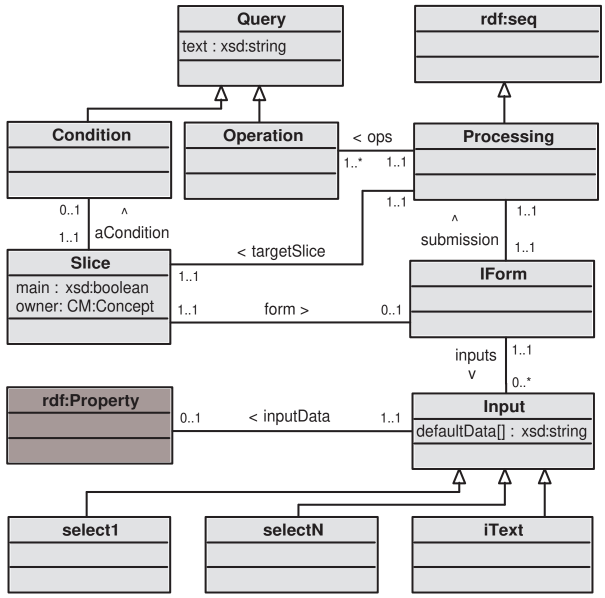

Figure 10. The extension of the application meta-model for dynamic web applications

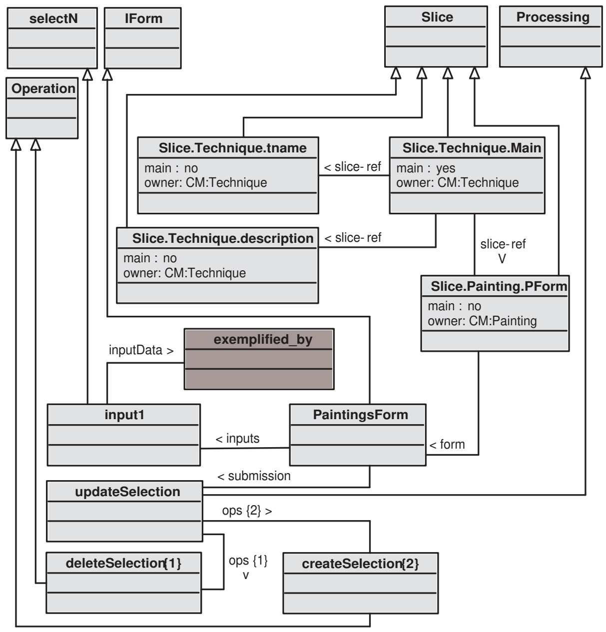

Figure 11. Implementation of slice Technique.Main in the extended application meta-model

manipulation) query. The `Processing` is a sub-class of the `rdf:seq` sequence. The `targetSlice` property defines to what slice to navigate after the processing is executed.

The implementation of the `Technique.main` slice using the extended meta-model is in Figure 11. It has two attributes `Slice.Technique.tname` and `Slice.Technique.description` implemented as slices, and the form `PaintingsForm` also encapsulated in the `Slice.Painting.PForm` slice. The form processing `updateSelection` is a sequence of two operations `deleteSelection` and `createSelection`, where the first is a data manipulation query that deletes all instances of the `inSelection` and `selPaintings` CM properties, and the second one creates new instances of the properties corresponding to the new selection.

5 Implementation

The Hera project includes also the development of prototypes of software engines generating and running web applications. The first type of engine (called HPG: Hera Presentation Generator) generates static web presentations (with a static AM as described in Section 4.1). Figure 12 shows the main window that allows to configure the complete setup of the generated presentation (selection of concrete AM, source data, user profiles, end format, etc.). It also allows stepwise processing (controlled by the user). The transformation process is based on XSLT processing using XSL stylesheets.

The engine producing dynamic web applications (with the dynamic AM as described in Section 4.2) runs as a servlet under a web server (Apache Tomcat). It calculates application pages on demand and allows processing of user

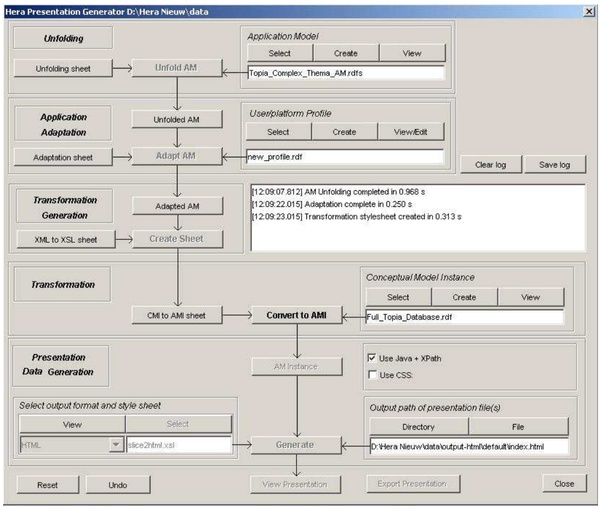

Figure 12. The main window of HPG

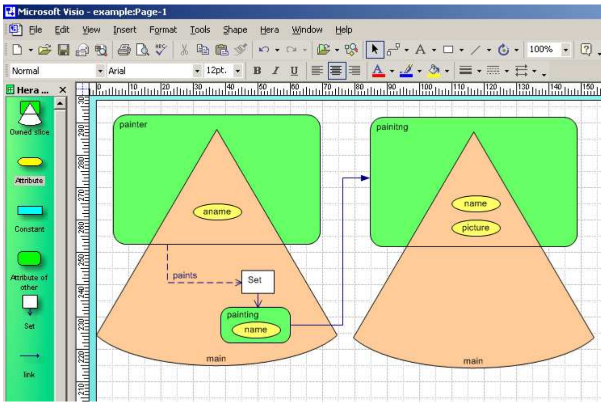

Figure 13. AM builder

inputs via HTML forms. When a form is submitted, the engine performs operations associated with the form (specified in AM), and then it creates the page corresponding to the target slice specified with form processing. When an http request for the next page (encoded in links or form submission elements as slice identifiers) is received, the engine according to the specifications in AM retrieves the required data and creates AMI for a certain page, that is then transformed into HTML. The engine is written in Java, uses HP Jena RDF API, and the Sesame repository.

Tools facilitating the design of Hera models are also under developmnet. They allow the designer to specify the models by drawing their graphical notation, what makes the design process more effective. Our tools translate the graphical notation into RDFS (OWL not yet) descriptions that are used by the aforementioned Hera engines. Concretely, we have implemented CM and AM builders (see Figure 13) that are based on Microsoft Visio templates.

6 Conclusion

In this paper we have addressed the role of semantical descriptions in the process in which in the Hera methodology web applications are designed. Hera models rely on the use of languages like RDF(S), OWL, and SeRQL to define how the navigation and personalization are specified for data-intensive web applications. The languages are flexible and extensible (allow schema refinement and enrichment, for instance the cardinality of properties that we use for CM in RDFS), and have explicit semantics (unlike e.g. XML) what allows us effectively use third party RDF software for the most of processing.

Meta-models expressed in the semantic web languages formalize the Hera methodology and precisely describe hierarchies of concepts (building blocks of models) that can be used in concrete models. Thanks to constraints specified within the meta-models, it is possible to automatically check the correctness of models, what can be also used in the designer tools already at design time.

References

[1] Aidministrator Nederland b.v. The serql query language. http://sesame.aidministrator.nl/publications/users/ch05.html.

[2] V. Balasubramanian, M. Bieber, and T. Isakowitz. A case study in systematic hypermedia design. *Information Systems*, 26(4):295–320, 2001.

[3] P. D. Bra, A. Aerts, G. J. Houben, and H. Wu. Making general-purpose adaptive hypermedia work. In *WebNet 2000 World Conference on the WWW and Internet*, pages 117–123. AACE, 2000.

[4] D. Brickley and R. V. Guha. Rdf vocabulary description language 1.0: Rdf schema. W3C Recommandation 10 February 2004.

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

[6] J. Gomez, C. Cachero, and O. Pastor. On conceptual modeling of device-independent web applications: Towards a web engineering approach. *IEEE Multimedia*, 8(2):26–39, 2001.

[7] N. Koch, A. Kraus, and R. Hennicker. The authoring process of the uml-based web engineering approach. In *First International Workshop on Web-Oriented Software Technology*, 2001.

[8] O. Lassila and R. R. Swick. Resource description framework (rdf) model and syntax specification. W3C Recommendation 22 February 1999.

[9] D. L. McGuinness and F. van Harmelen. Owl web ontology language overview. W3C Recommendation 10 February 2004.

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

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