Modeling User Input and Hypermedia Dynamics in Hera

Geert-Jan Houben, Flavius Frasincar, Peter Barna, and Richard Vdovjak

Technische Universiteit Eindhoven
PO Box 513, NL-5600 MB Eindhoven, The Netherlands
{houben, flaviusf, pbarna, rvdovjak}@win.tue.nl

Abstract. Methodologies for the engineering of Web applications typically provide models that drive the generation of the hypermedia navigation structure in the application. Most of these methodologies and their models consider link following as the only materialization of the navigation structure. In this paper we see how extended user input can dynamically influence the navigation structure. By means of Hera it is shown how one can define this extended user input and capture the functional aspects related to the hypermedia dynamics in the RDF(S)-based design models. For this purpose we discuss the definition of input controls, the representation of state information, and the embedding of both in the application model. We also present the XML/RDF-based architecture implementing this.

1 Introduction

Under the influence of the World Wide Web we have seen the development of a new type of (data-intensive) information systems. These so-called Web Information Systems (WIS) [1] are characterized by the use of hypermedia navigation through the content of the system, in combination with the traditional functions of an information system allowing to update and query the content. As examples of WIS applications we mention online services like real-estate sales, employee information, museum information, or mail order catalogs.

The engineering of WIS requires different methodologies than the ones than we have been using for information system development over the last decades. In the traditional approach, used for example in more database-oriented applications, we see that most of the engineering activity is related to structuring the data so that the structure matches the standard software component, i.e. relational database. The subsequent design of presenting the content to the user is considered in the query facility associated with the software. On the other hand, with the original hypermedia approaches we see a different pattern, since they typically assume a process of manually linking documents. The design process centers on the design of the navigation in the presentation of the content in terms of a hyperdocument.

In the engineering of a WIS the designer has a challenging task. On the one hand, the designer has to provide the users with all benefits from using the hypermedia paradigm and particularly the notion of navigation through the information offered by the system. On the other hand, the designer has to support the users in their maintenance of the content by allowing updates and queries to the data. Many of today's data-intensive Web applications show the designer's attention for the maintenance of the data, but at the same time they show the risk of losing those benefits of hypermedia that have been the foundation for the success of the Web.

In the research field of WIS engineering we have seen proposals for methodologies that extend and improve the methodologies for manual hypermedia design for application in data-intensive information systems: we mention as representatives RMM [2], WebML [3], OOHDM[4], OOWS [5], UWE [6], OO-H [7], and Hera [8]. Typically these methodologies distinguish themselves from standard information system development methodologies by their explicit attention for the navigation design. Since however the WIS applications contain content that is highly dynamic, the design has to support the dynamics involved with the content. This support includes not just updating the content stored in the system, but also allowing the user to affect the hypermedia presentation of the data. Illustrative examples of this influence of the user on the hypermedia presentation are the history facility that allows the user to go back outside of the presented hyperlinks, or the shopping basket concept that allows the user to store some information temporarily during a browsing session. Such influence implies that a certain "state" is stored by the system to allow the user to interact with the hypermedia presentation and particularly with its navigational structure.

As we indicated earlier the available WIS engineering methodologies have a strong focus on the generation of navigation over the content. The user's actions consisted of following links, and as a consequence all the system could do was based on that. The history facility is a straightforward example. Giving the user more possibilities to interact with the generated hypermedia presentation can help to define or limit the hyperspace and thus to realize personalization and adaptivity. In a museum application asking the user to define what is interesting for him can help the system to create a more suitable navigation structure with specific information about those items on display that interest the user/visitor. Another example of user influence would be the role of a shopping basket in the sales communication based on a product catalog; not so much for the registration of the sales order, but certainly for the adjustment of the presentation in accordance with the user input, for example by showing a page with the complete contents of the shopping basket (order). As one of the consequences of this extended user influence there is a need to deal with navigation data, i.e. data primarily there to support the user in influencing (e. g. restricting, selecting) the navigation view over the application domain data. In this paper we show how to model this dynamic navigation through Hera models that allow the specification of the extended user input, the management of navigation information, and the effect of both of them on the hypermedia presentation. For this specification Hera uses semantic web languages that are very suitable for modeling of semi-structured data and describing their semantics.

In Section 2 we discuss how related work supports this kind of extended user input in relation to hypermedia dynamics. Section 3 highlights the main principles of the Hera approach, before we discuss in Section 4 the details of extended user input and dynamics in Hera: first we present the input controls, then the navigation data model, its effect on the application dynamics, and finally the architecture of the implementation.

In the conclusion we name the main advantages of this approach compared to other approaches.

2 Related Work

In this section we take a closer look at two well-known representative methodologies WebML and OOHDM to see how they support modeling of user input and hypermedia (navigation) dynamics.

In WebML [3] the page content and navigation structure is captured in the (advanced) hypertext model using a predefined set of modeling primitives. The infrastructure for user input consists of data entry units that have associated with them operation units. A data entry unit contains a set of input fields that can be filled by users or can have default values. Data entry units have one or more outgoing links that are activated when the user fills input fields and submits the information. With a link can be associated parameters that transfer the input values to the destination unit(s), for example for further processing by an operation unit. There are several predefined operation units, for instance for activating external web services or content management operations like creation, deletion, and update of entities and relations. The whole library of units is open (new units can be defined in XML) and contains a number of data entry units for different kinds of user inputs. All contextual information passed between the units by link parameters is described in separate XML files.

The user input is in OOHDM [4] specified by means of interface objects that are defined on top of the navigation structure specification. The navigation is described using navigation classes derived from concept classes, navigation contexts representing collections of navigation objects, and access structures like links, indices, or guided tours. The interface objects are instances of interface classes expressed by Abstract Data Views (ADV). Every ADV defines a set of events (triggered by users) it can handle via methods of navigational classes, and a set of attributes that can be perceived by users. The processing of user-triggered events is specified in ADV charts, where the events are mapped to messages that are sent to navigation objects and can change their state.

3 Hera Methodology

The Hera methodology [8, 9] is a model-driven methodology for designing WIS. Before we concentrate on user input and hypermedia dynamics in the next section, we will briefly describe the main aspects of Hera. In response to a user query a WIS will gather (multimedia) data possibly coming from heterogeneous sources and will produce a meaningful hypermedia (Web) presentation for the retrieved data. The Hera methodology automates such a process by providing high level abstractions (in terms of models) that will drive the (semi-)automatic presentation generation. Moreover, Hera enables the presentation adaptation based on user preferences and device capabilities, which means that the presentation generation takes into account issues like the platform being used (e. g. PC, PDA, WAP phone) [10].

Based on the principle of separation of concerns and for the sake of interoperability several models have been distinguished. Because these models are considered Web metadata descriptions that specify different aspects of a WIS, we chose to use the Web metadata language, i.e. RDF(S) [11, 12], to represent all models and their instances. Our choice is also justified by the RDF(S) extensibility and flexibility properties that enabled us to extend the language with model specific primitives to achieve the desired power of expression. As RDF(S) doesn't impose a strict data typing mechanism it proved to be very useful in dealing with semistructured (Web) data.

The Hera toolset implements this methodology by offering software for the automatic generation of hypermedia based on the different Hera models. In order to facilitate the building (and visualizing) of these models, several Visio solutions were implemented. Such solution is composed of a stencil that will display all the model shapes, a drawing template, and a load/export feature providing the RDF(S) serialization of Hera models. Throughout the paper we use a running example based on the metadata associated to about 1000 objects from the Rijksmuseum. Figure 1 depicts (in the CM Builder) a part of the CM for our example, while Figure 2 illustrates the corresponding AM.

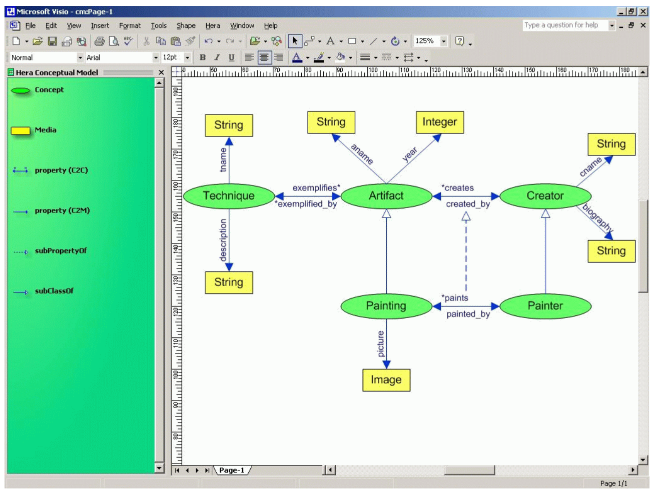


Fig. 1. Conceptual model

The conceptual model (CM) describes the structure (schema) of the application domain data. This structure is described using RDFS in terms of concepts and concept relationships. A concept has attributes, i.e. properties that refer to some media instances. For concept relationships we define their cardinalities and their inverse relationships.

The application model (AM) specifies the structure of navigational view over the application domain data. This structure is also defined using RDFS, where the hypermedia presentation is described in terms of slices and slice relationships (inspired by RMM). A slice is a meaningful presentation unit that groups concept attributes (from CM) that need to be presented together on the user display. There are two types of slice relationships: compositional relationships (for embedding a slice into another slice) and navigational relationships (as hyperlink abstractions).

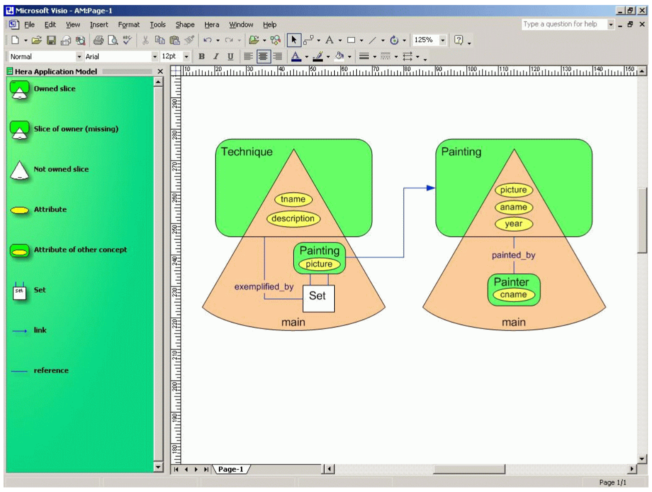


Fig. 2. Application model

4 User Input and Hypermedia Dynamics in Hera

In most WIS design methodologies, the only kind of interaction considered is link following: the use of the navigation structure is equivalent to wandering through the structure by clicking on anchors and following links. In our extended approach we go a step further and consider other forms of user input and dynamics with respect to this hyperstructure. Therefore, in the next subsections we describe:

– information for navigation dynamics, defined in the navigation data model
– user input controls with associated processing of navigation information
– application model extended with the user input
– architecture of a Hera system

We illustrate this by an example from our museum application that allows the visitor to buy posters of the paintings in the museum.

4.1 Navigation data model

In addition to the data in the aforementioned models CM and AM, 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 a so-called navigation data model (NDM). The purpose of this model 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 NDM of our example is depicted in Figure 3; it consists of the following concepts:

- The $SelectedPainting$ concept is a subclass of the $Painting$ concept from the CM. It represents those paintings which the user selected from the multi-selection form.
- 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 $contains$.

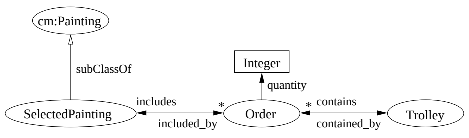


Fig. 3. Navigation data model

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 instantiation of both groups of concepts is triggered by a certain action (an acknowledgement such as pressing the submit button) specified in the AM. Each such action can have an associated query which either defines the view (the first group) or specifies what instances should be inserted in the concept's extent (instantiation). The data resulting from the query execution is represented in the NDM instance (NDMI) and stored as state information till the next change (query) occurs[1]. The AM can refer to the concepts from NDM as if they were representing real data concepts.

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 runtime. This is reflected also in the NDMI depicted in Figure 4 that results from the user's desire to buy 3 posters of the selected painting. The instance $Painting1$ comes 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 4. Note that for presentation purposes (backwards link generation) we also generate for every property its inverse.

[1] We can see an entire spectrum, going from updating the content to just using state data to help change the hypermedia structure. In this paper we focus on the state data that helps specifying the interaction with the navigation structure (since updating the content is possible but outside presentation generation).

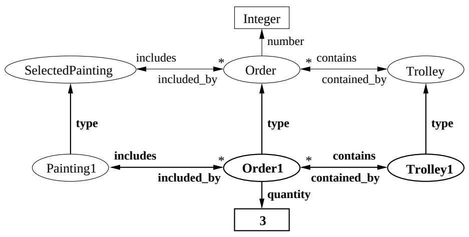
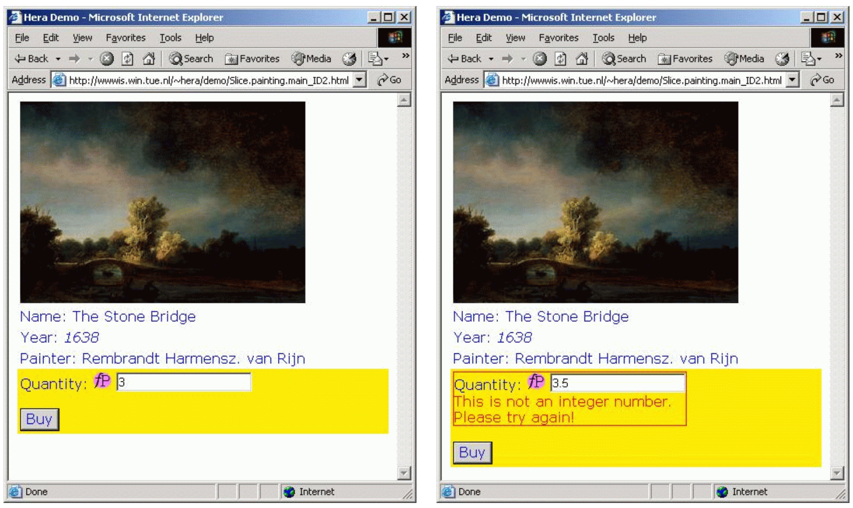


Fig. 4. Navigation data model instance

4.2 Input controls

In Figure 5(a) we see from the implementation a slice of a painting selected by the user. It shows that in this slice the user is provided with a form to enter a quantity that represents the number of posters of this painting that the user considers to buy. In Figure 5(b) we see another example where the form is instructed to respond to the user's attempt to enter a non-integer value.


Hera Demo - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go
Done
Internet

http://wwwis.win.tue.nl/~hera/demo/Slice.painting.main_ID2.html

Name: The Stone Bridge
Year: *1638*
Painter: Rembrandt Harmensz. van Rijn
Quantity: 3
Buy

http://wwwis.win.tue.nl/~hera/demo/Slice.painting.main_ID2.html

Name: The Stone Bridge
Year: *1638*
Painter: Rembrandt Harmensz. van Rijn
Quantity: 3.5
This is not an integer number.
Please try again!
Buy

(a) Good input (b) Bad input

Fig. 5. Form with input in browser

In the Hera software we implemented the user input forms using the XForms [13] standard. As an XForm processor we used formsPlayer [14], a plug-in for Internet Explorer[2]. In defining application forms we were inspired by XForms' clean separation of data from controls.

For these forms we need primitives in the AM that specify the functional embedding of the controls in the navigation structure. Figure 6 shows three examples of how we specify the embedding of controls in AM. In the leftmost example, the $SelectForm$ allows to make a choice for multiple items out of a list of paintings. The AM primitive shows the concept that "owns" the form, in this case $Painting$; it shows the items that are displayed in the form, in this case names; finally, it shows the items that are handed over by the form to the subsequent navigation: the name of the selected painting. In the middle example the form is similar but allows a choice of exactly one out of multiple options. The rightmost example shows a form called $BuyForm$ that allows user input, in this case to enter the quantity of posters considering to buy. The form hands over the tuple consisting of the entered quantity and the painting name (the painting information is taken from the form's context).

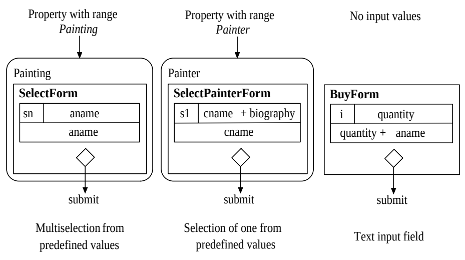


Fig. 6. Forms in AM

So we see that for the user input controls we specify in the diagram for the AM the relevant parameters that make up the form. Thus we describe the relevant functional aspects of the form, and are able to abstract in the diagram from the actual form code. Similar to XForms we distinguish between the input controls and their state information stored in separate models.

Figure 7 presents the models for the forms $SelectForm$ and $BuyForm$. It consists of two form types, $Form1$ defines the type of the $SelectForm$ and $Form2$ defines the type of the $BuyForm$. A Hera form model instance represented in RDF/XML corresponds to the associated XForms model instance. The $Integer$ type matches the XML Schema [15, 16] type $xsd{:}integer$ and the $String$ type matches the XML Schema type $xsd{:}string$. In case that the user enters a value of a different type than the one specified in the form model, an XForms implementation (see Figure 5(b)) will immediately react with an error message (due to its strong type enforcement capabilities).

Figure 8 describes two possible model instances for the form models given in Figure 7. In the $SelectForm$ the user selected two paintings and in the $BuyForm$ the user decided to buy one of these paintings.

[2] The small logo labelled "fP" in Figure 5(a) is the $f$orms$P$layer signature in the implementation.

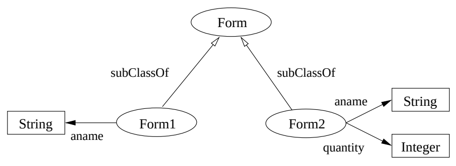


Fig. 7. Form models

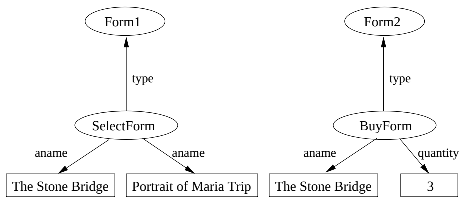


Fig. 8. Form model instances

4.3 Application model

With the aid of the aforementioned primitives we are able to express the user input in our museum example in terms of an (extended) AM. Figure 9 depicts the part of the AM which captures the user input.

The $Technique$ slice contains a form that lists all paintings exemplifying that particular technique and offers the user the possibility to select some of these paintings. For the latter we see in the $Technique$ slice the input control called $SelectForm$ with $Painting$ as its owning concept (meaning that this form is selecting $Painting$ concepts). We also see that the form lists the paintings by their $aname$ property and produces for each selected painting the $aname$ property to identify the selected paintings.

After selecting a painting, the outgoing slice navigational relationship denotes that the form in the $Technique$ slice results in navigation to a slice that represents the set of selected slices, each represented by their $aname$, and also in a $Trolley$ that, while initially empty, will contain the paintings that actually are going to be bought. A $Trolley$ contains a set of $Order$s, while an $Order$ represents the request to buy a poster of a (selected) painting in a certain quantity.

The navigation can go further to the $SelectedPainting$ slice. That slice includes not only all the properties that represent the painting, but also a form called $BuyForm$ with a user input control. That control allows the user to specify the quantity (of posters of this painting to buy). After filling this form the user can navigate via the outgoing slice relationship to the next slice where the trolley is maintained (and where the user can decide to select another painting for considering in more detail).

With these slices and slice navigational relationships in the AM we have specified the entire navigation structure. In the AM diagram we exploit the fact that the functionality of the controls is standard, e. g. the selection of n items from a list; therefore the diagram only indicates which standard control is used. What we also do indicate is the signature: we give the properties displayed in the form, and the identification of

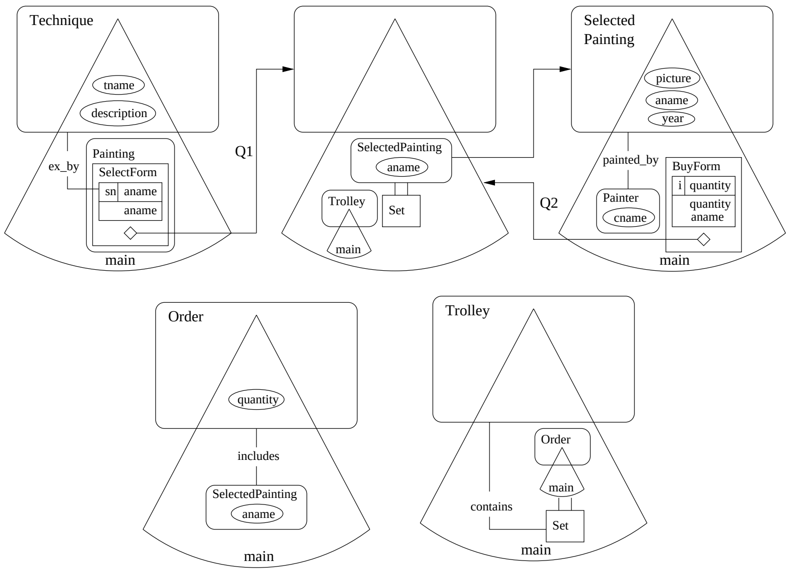


Fig. 9. Extended application model

the concepts forwarded via the slice navigational relationship. Note that both slice navigational relationships that emerge from the forms ($Q1$ and $Q2$) are in fact queries. In the query definition we will use the prefix cm: for concepts/properties coming from the conceptual model, the prefix ndm: for concepts/properties specified in the navigation data model, and $form$: for concepts/properties introduced in the form model.

The RDF model instance of $SelectForm$ is given in Figure 10. The query $Q1$ creates a view over painting instances from the CM instance (CMI) which were selected by the user in $SelectForm$. This view defines the instances of the ndm:$SelectedPainting$ class from the NDM.

```
<Form1 rdf:ID="SelectForm">
    <aname>The Stone Bridge</aname>
    <aname>Portrait of Maria Trip</aname>
</Form1>
```

Fig. 10. Model instance for SelectForm

Figure 11 describes this query in the SeRQL [17] notation. The actual form is modelled as an RDF resource with multiple $form$:$aname$ properties containing the names (values) of those paintings which were selected by the user.

The RDF model instance of $BuyForm$ is given in Figure 12. The query $Q2$ associated with the $BuyForm$ creates a new instance of the NDM concept ndm:$Order$ each time the user decides to buy a poster of a selected painting.

```
CONSTRUCT
    {P}<rdf:type>{<ndm:SelectedPainting>}
FROM
    {P}<rdf:type>{<cm:Painting>};
       <cm:aname>{Paname}
WHERE
    Paname IN SELECT Faname
              FROM   {SF}<form:aname>{Faname},
                     {SF}<rdf:type>{<form:Form1>},
                     {SF}<rdf:ID>{Fname}
              WHERE Fname = "SelectForm"
```

Fig. 11. User query Q1

```
<Form2 rdf:ID="BuyForm">
    <aname>The Stone Bridge</aname>
    <quantity>3</quantity>
</Form2>
```

Fig. 12. Model instance for BuyForm

The SeRQL translation of this query is presented in Figure 13. The form is modeled similarly as before by an RDF resource with two properties $form{:}quantity$ and $form{:}aname$. Note that the (old) instance of the $ndm{:}Trolley$ exists outside this form and is created beforehand during the initialization of the session.

```
CONSTRUCT
    {O}<rdf:type>{<ndm:Order>};
       <ndm:quantity>{Fquantity};
       <ndm:includes>{Fpainting},
    {T}<ndm:contains>{O}
FROM
    {T}<rdf:type>{<ndm:Trolley>},
    {Fpainting}<cm:aname>{Paname};
               <rdf:type>{<ndm:SelectedPainting>},
    {BF}<form:quantity>{Fquantity};
        <form:aname>{Faname},
    {BF}<rdf:type>{<form:Form2>},
    {BF}<rdf:ID>{Fname}
WHERE
    Paname = Faname AND
    Fname = "BuyForm"
```

Fig. 13. User query Q2

4.4 Architecture

While in the previous subsections we have paid attention to the specification of user input and hypermedia dynamics and the way in which the AM can support this extended interaction specification, we now turn to the implementation. As we have indicated earlier the software of the Hera toolset can generate the hypermedia structure from the given models. In other work [8] we have sketched the (software) architecture for the case of normal link following. In Figure 14 we see how we extended the architecture such that it supports the handling of user input, e. g. via the forms[3].

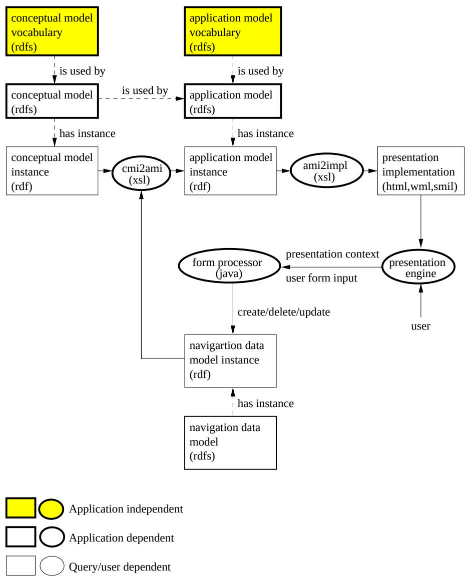


Fig. 14. Presentation generation

We see that the $presentation\ engine$ is responsible for serving the generated presentation to the user. As soon as the engine discovers user input it hands this over to the $form\ processor$ that is going to interpret the actual user input (and possibly the contextual information that explains with what the user input is associated). So, the $form\ processor$ can get the quantity of posters to buy and the context that explains what the painting is for which the user wants to buy this quantity. The $form\ processor$ can produce data that is added to the NDMI. The information from NDMI and CMI is used together in order to generate/update the AM instance.

As we have indicated in Figure 14 the implementation fits nicely in the RDF-based approach that we already had for the link following. By adding the additional model information in RDF(S) we can perfectly manage the additional functionality of explicit user input resulting in a different hypermedia presentation. We implemented data transformations (see $cmi2ami$ and $ami2impl$ in the figure) by means of XSLT stylesheets [18]. This was made possible due to the XML serialization of RDF model instances and the fact that XForms [13] is XML-based. Besides its XML interoperability XForms offers also device independence, which enables to use the same form on multiple platforms. X-Smiles [19] provides a good view on how the same XForms will look like on desktop, PDA, WAP phone etc. As an XSLT processor we used Saxon [20]

[3] We have focussed here on the part of the architecture for extended user interaction, and left out the architecture description for the rest of the software.

which implements XSLT 2.0 and XPath 2.0. In the form processor we employed the Java-based Sesame [21] implementation of the SeRQL query language [17].

5 Conclusion

By providing new primitives in Hera, e. g. for capturing the user input, it is possible to considerably extend the class of applications that can be specified. As an example we mention the use of primitives like the “shopping basket” or “list selection” that are so typical for applications in the context of services provided via the Web. Another extension is the increased support of dynamics. With the new primitives and the navigation data model it has become possible to handle effectively the dynamic adaptivity known from adaptive hypermedia [22]. For example, for such personalization purposes the navigation data model can store the necessary user model (both its temporary and persistent parts), while the application model can specify the necessary adaptation rules. The construction of the navigation view over data can be further enhanced by existing generic methods for the development of navigation structures based on user interaction modeling, for instance [23].

Comparing these new facilities in Hera to other work, we see that the explicit support of input controls such as forms are not modelled explicitly in OOHDM for example. It does distinguish mouse-related aspects of user input, but compared to Hera it does have a more limited support of data-entry by the user. This aspect of user input is a strong point of WebML, but in fact that facility is more concerned with the content management of the information system. In Hera this is also supported, at the level of the conceptual model, but besides of this Hera allows to combine the stored data in the conceptual model with the auxiliary navigation data stored in the navigation data model. WebML has, next to the link parameters, also global parameters that can model a “state” in terms of attribute-value pairs, but Hera goes much further in the specification of this state information allowing also complex relationships between concepts (represented in graphs).

The fact that Hera uses RDF(S) representations of the models gives a number of advantages over other approaches. To start with, it supports the semistructured data that is so typical for the Web. With RDF(S) Hera offers increased interoperability, e. g. for the exchange and sharing of user models. It also allows to express complex queries (e. g. in the design of the dynamics) that make use of the subclassing mechanism. Moreover, the modeling in Hera of the extended user input inherits good principles from existing standard like XForms. It chooses to separate in the user input the controls (the presentation aspects) from their models (the data aspects). Opposed to other approaches, Hera has a concrete implementation based on these standards.

In future we plan to incorporate the possibility of web service invocation within applications designed using Hera on different levels: on the conceptual (data) level (web services will act as virtual instances of data concepts), and on the application level (web services will provide building blocks of slices). Furthermore, we investigate general properties, mutual relationships, and constraints of Hera models that would help us to build tools for the automated checking of correctness of the models.

References

1. Isakowitz, T., Bieber, M., Vitali, F.: Web information systems. Communications of the ACM **41** (1998) 78–80
2. Balasubramanian, V., Bieber, M., Isakowitz, T.: A case study in systematic hypermedia design. Information Systems **26** (2001) 295–320
3. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann (2003)
4. Schwabe, D., Rossi, G.: An object oriented approach to web-based application design. Theory and Practice of Object Systems **4** (1998) 207–225
5. Pastor, O., Fons, J., Pelechano, V.: Oows: A method to develop web applications from web-oriented conceptual models. In: International Workshop on Web Oriented Software Technology (IWWOST). (2003) 65–70
6. Koch, N., Kraus, A., Hennicker, R.: The authoring process of the uml-based web engineering approach. In: First International Workshop on Web-Oriented Software Technology. (2001)
7. Gomez, J., Cachero, C. In: OO-H Method: extending UML to model web interfaces. Idea Group Publishing (2003) 144–173
8. Vdovjak, R., Frasincar, F., Houben, G.J., Barna, P.: Engineering semantic web information systems in hera. Journal of Web Engineering **2** (2003) 3–26
9. Frasincar, F., Houben, G.J., Vdovjak, R.: Specification framework for engineering adaptive web applications. In: The Eleventh International World Wide Web Conference, Web Engineering Track. (2002) `http://www2002.org/CDROM/alternate/682/`.
10. Frasincar, F., Houben, G.J.: Hypermedia presentation adaptation on the semantic web. In: Adaptive Hypermedia and Adaptive Web-Based Systems, Second International Conference, AH 2002. Volume 2347 of Lecture Notes in Computer Science., Springer (2002) 133–142
11. Brickley, D., Guha, R.V.: Rdf vocabulary description language 1.0: Rdf schema. (W3C Working Draft 10 October 2003)
12. Lassila, O., Swick, R.R.: Resource description framework (rdf) model and syntax specification. (W3C Recommendation 22 February 1999)
13. Dubinko, M., Klotz, L.L., Merrick, R., Raman, T.V.: Xforms 1.0. (W3C Recommendation 14 October 2003)
14. x-port.net Ltd.: (formsPlayer) `http://www.formsplayer.com`.
15. Biron, P.V., Malhotra, A.: Xml schema part 2: Datatypes. (W3C Recommendation 02 May 2001)
16. Thompson, H.S., Beech, D., Maloney, M., Mendelsohn, N.: Xml schema part 1: Structures. (W3C Recommendation 02 May 2001)
17. Aidministrator Nederland b.v.: (The serql query language) `http://sesame.aidministrator.nl/publications/users/ch05.html`.
18. Kay, M.: Xsl transformations (xslt) version 2.0. (W3C Working Draft 12 November 2003)
19. X-Smiles.org et.al.: (X-Smiles) `http://www.x-smiles.org`.
20. Kay, M.: (Saxon) `http://saxon.sourceforge.net`.
21. Aidministrator Nederland b.v.: (Sesame) `http://sesame.aidministrator.nl`.
22. Bra, P.D., Houben, G.J., Wu, H.: Aham: A dexter-based reference model for adaptive hypermedia. In: The 10th ACM Conference on Hypertext and Hypermedia, ACM (1999) 147–156
23. Schewe, K.D., Thalheim, B.: Modeling interaction and media objects. In: Natural Language Processing and Information Systems. Volume 1959 of Lecture Notes in Computer Science., Springer (2001) 313–324