User Interaction in Modern Web Information Systems

Peter Barna - Geert-Jan Houben
Technische Universiteit Eindhoven
PO Box 513, NL-5600 MB Eindhoven, The Netherlands
{p.barna, g.j.houben}@tue.nl

Abstract

Modern Information Systems based on Web technologies (Web-based Information Systems - WIS) typically generate hypermedia presentations according to the user needs. Hera is our model-driven methodology specifying the design cycle and the architecture framework for WIS. To avoid additional expensive programming the functionality of generated hypermedia presentations is limited to following links. However, modern e-commerce applications require more sophisticated user interaction. In this paper we discuss extensions of the Hera methodology regarding the design of such interactive WIS. We explain the main ideas of the extension on the example of a virtual shopping cart application, as a typical pattern appearing in e-commerce applications.

1. Introduction

Many information systems today use the Web as a platform. Their remote clients interact with the system through Web browsers. Increasing demands of E-commerce require richer functionality of such Web-based Information Systems (WIS). This rich functionality goes together with richer means of user-interaction compared to just following links. For the sake of conciseness in the rest of the paper we call such WIS (though not completely correct) *interactive WIS.*

Due to the specific nature of the Web, a number of methodologies have been developed particulary for WIS design ranging from earlier methodologies as RMM [8], through the object-oriented approaches as OOHDM [13] and UWE [11], to methodologies as WebML [2]. Some of the methodologies do not support the design of interactive WIS (e.g. RMM considers only static navigation specification), and some do (e.g. object-oriented methods or WebML).

In our perception, a WIS is an information system generating hypermedia presentations delivered by means of the Web to users. Hera [5] is a model-driven WIS design methodology that specifies a number of design steps and their outputs in terms of models. The models describe different facets of the system and are used during the process of hypermedia presentation generation. Every concrete model is built from primitive concepts that are defined and hierarchically organized in a *schema* for the model. An analogical example in an object-oriented structure modelling method (e.g. UML class diagram) is a concrete class structure where a schema for these models defines terms as "class", "association", "specialization", etc. and their relationships.

Hera supports the generation of adaptable and adaptive hypertext presentations. Adaptability is adjusting presentations to features known before the generation process (for example the characteristics of the hardware platform: a presentation looks differently on a WAP phone and on a PC). Adaptivity is conditional inclusion of page fragments and conditional link hiding where both are based on dynamically changing features: a user model is dynamically updated during the browsing. Although both mechanisms contribute to the usability of presentations for concrete users, they do not make presentations interactive (in the sense we have defined earlier).

Even though Hera was not explicitly aimed for the design of interactive WIS, we consider possibilities of its deployment for these applications. In this paper we investigate the application of Hera for authoring of interactive WIS. A possible application is demonstrated on a simple example of a poster sales process in an on-line museum poster shop, where we show the combination of specifying interaction and navigation structure.

In section 2 basic principles of Hera's methodology and framework are briefly described. The requirements for the on-line poster shop example are specified in section 3, and the proposed design focusing on the navigation and interaction aspects is explained in section 4. Possible consequences for the extension of the Hera architecture and its implementation are discussed in section 5.

2. Hera

In Hera a WIS accepts a request from the user, collects necessary data from distributed data sources, integrates the data, and using a process of data transformations forms a hypermedia presentation in concrete end format (e.g. HTML).

The Hera methodology defines a set of design steps that need to be taken to build a set of models and data transformation procedures. The models specify views on certain aspects of the transformation process, particulary the structure of data in different stages of the process. Concrete models are constructed from primitive concepts that are defined in so called model schemas.

According to Hera model data processed in a WIS is in the RDF (Resource Description Framework) [12] format serialized in XML. The benefit of using RDF is its more explicit semantics compared to XML. For definitions of models and model schemas we use RDF Schema (RDFS) [1]. For query specifications within the system we use RDF Query Language RQL [9].

2.1 Methodology

Typical WIS design methodologies distinguish the following phases:

- **Requirement Analysis** gathers and forms the specification of the user requirements.
- **Conceptual Design** defines the conceptual model for the problem domain.
- **Navigation Design** builds the navigation view of the application.
- Some methodologies include **adaptation design**, where the adaptation model is built and all associated mechanisms are defined.
- **Presentation Design** defines the appearance and layout of generated presentations.
- **Implementation** realizes the WIS itself.

The Hera methodology redefines the following phases and models (see Figure 1):

- **Conceptual and integration design**. The main outputs of this phase are the Conceptual and Integration Models (CM, IM). Since we consider a distributed and heterogeneous (in format and content) data repository, CM gives a unified semantic view on the repository (to facilitate further design steps), whereas IM specifies semantic links from concepts in particular sources to concepts in CM.
- **Application and adaptation design**. In this phase the designer creates the Application Model (AM), and a set of models for adaptation (e.g. set of adaptivity rules for adaptivity, a specification of the user/platform profile, and initial user model). AM is built on top of CM and describes the overall structure of generated presentations including navigation. Adaptation of the generated presentation is based on static features (known before the generation process, e.g. user/platform properties), or the dynamic features (changing during browsing) based on a user model. For adaptivity we can use the AHA! engine [3] (based on AHAM reference model [4]) is used.
- **Presentation design**. In this phase the designer specifies the layout and rendering of presentation units in the presentations.

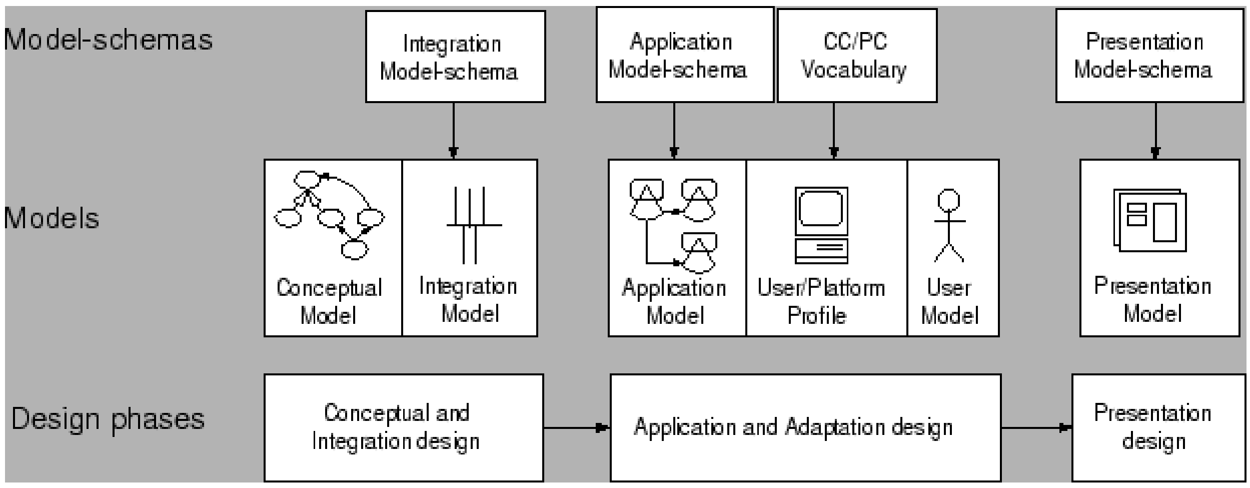

Figure 1: Design phases and models in Hera

2.2 Models

Models in Hera specify different facets (views) of the system and presentations generated by this system. In the following paragraphs we shortly explain models we use later in the example: conceptual and application models.

2.2.1 Conceptual Model

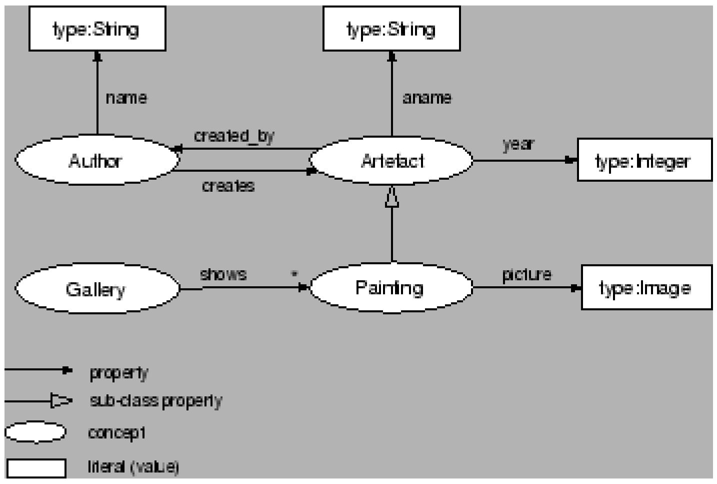

Figure 2: Example of a CM

The conceptual model describes the semantics of the data repository (problem domain) by means of concepts and their properties. The properties have as values other concepts, or concrete values (literals). There is a special property, the sub-class property, that represents concept inheritance. The schema for CM is actually a RDFS data model with added properties *cardinality* describing multiplicity of other properties, and *inverse* representing reversal of properties. An example of CM represented in RDFS graphical notation is in Figure 2.

The ovals represent concepts, and the rectangles represent literals (values). Concepts can have arbitrary properties with ranges of the types concept or literal. For the `sub-class` property the range concept inherits all properties of its domain concept, for instance, the `Painting` concept has also the `aname` property.

An example of inverse properties are the `created_by` and `creates` properties. An example of a property with multiple cardinality is the `shows` property (note the star).

2.2.2 Application Model

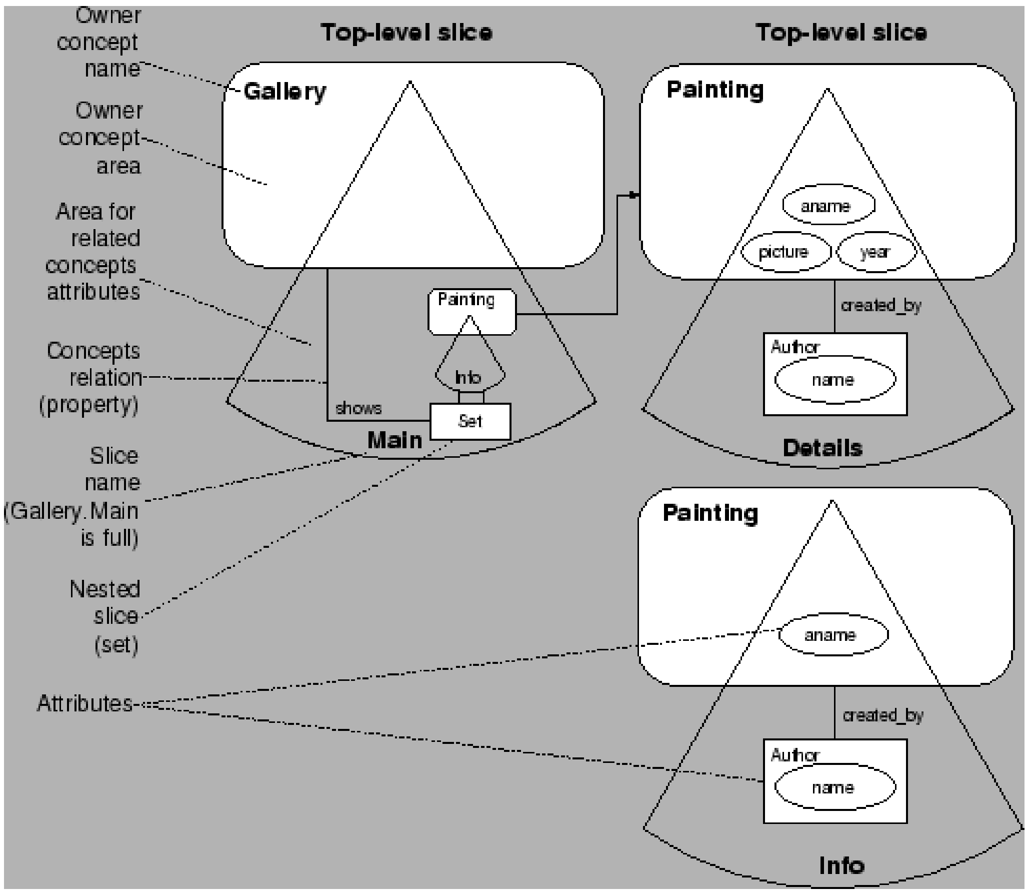

Figure 3: Example of an AM

The application model describes groupings of concept attributes (from CM) to semantically meaningful units and relationships between the units. Such units are called slices. A slice contains selected attributes of a so called owner concept, but can also contain attributes of related concepts. Slices can be aggregated (one slice can contain another slices), or referenced. Slices roughly represent page fragments in generated presentations (of course without spatial, temporal, or rendering details), and top-level slices (not contained in another slices) represent pages. References represent links.

An example of an AM based on the CM from the previous paragraph is in Figure 3. The top-level slice `Gallery.Main` has the nested slice `Painting.Info` that will be rendered as a set of its instances (the `shows` property in CM has multiple cardinality). From a concrete slice instance of `Painting.Info` there is a reference (link) to concrete instance of `Painting.Details` showing complete information about the given painting.

2.3 Data Transformations

When the user queries the data repository and wants to obtain the desired information in form of a hypermedia presentation, his query is re-distributed over data resources, and the data transformation process is performed in the steps (Figure 4):

- **Integration and data retrieval**, where the required data is collected from different data sources and transformed into a CM instance. The IM is used for the query re-distribution and data integration.
- **Presentation generation**, where the CM instance is transformed into an AM instance, and then into a final presentation in concrete format (e.g. HTML, WML, or SMIL) using PM. During generation of an AM instance, the adaptability conditions are evaluated and appropriate AM elements are included into the presentation. Moreover, the user model is instantiated.

All data transformation procedures in Hera are specified in XSLT [10] sheets. The procedures do not depend on concrete models. More details of Hera can be found in [5,7].

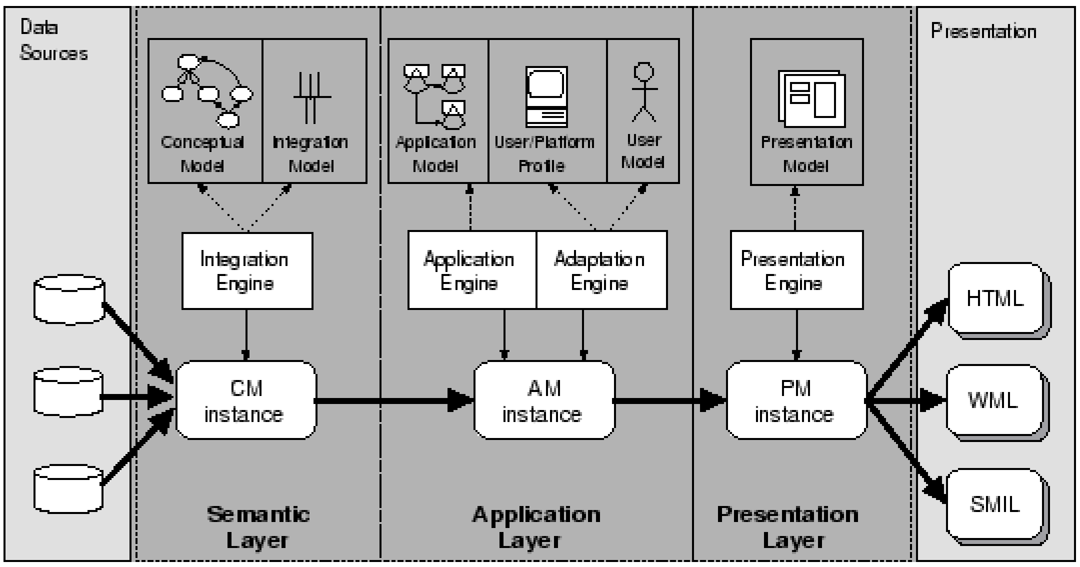

Figure 4: Data transformations in Hera perspective

3. Poster Shop Example

The example is an extension of a museum site with on-line shop selling posters related to paintings from the museum. Figure 5 shows details of how we envision the structure of application pages for the sales process, and specifically the shopping cart. The application should allow searching for desired posters based on their subjects. The user can put selected items into the shopping cart, can report and update the content of the cart, and finally can confirm the purchase.

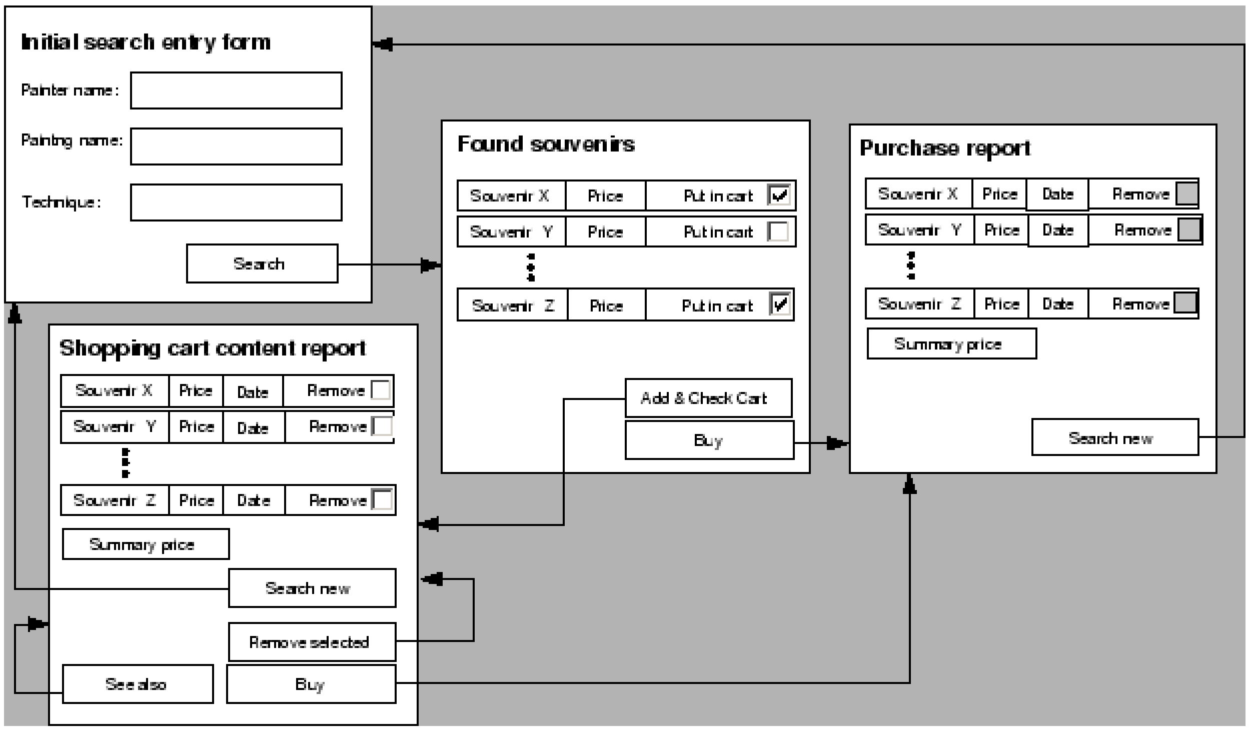
Initial search entry form
Painter name:
Painting name:
Technique:
Search
Found souvenirs
Souvenir X | Price | Put in cart
Souvenir Y | Price | Put in cart
Souvenir Z | Price | Put in cart
Add & Check Cart
Buy
Purchase report
Souvenir X | Price | Date | Remove
Souvenir Y | Price | Date | Remove
Souvenir Z | Price | Date | Remove
Summary price
Search new
Shopping cart content report
Souvenir X | Price | Date | Remove
Souvenir Y | Price | Date | Remove
Souvenir Z | Price | Date | Remove
Summary price
Search new
Remove selected
See also
Buy

Figure 5: Envisioned application structure

A typical user scenario is:

- The user sees the `Initial search entry form`, where he can enter names for a painting, painter, or technique to find posters related to paintings matching the criteria.
- By pressing the `Search` button the `Found posters` page is rendered with a list of posters matching entered criteria. The user marks the items he wants to put into his virtual shopping cart. Marked items can be bought by pressing the `Buy` button, or the content of user's shopping cart can be

viewed and updated by pressing the `Add & Check Cart` button.
- If the content of the cart was bought, the `Purchase report` page is displayed with the option of a new search.
- If the `Shopping cart content report` was displayed, the user can remove items from it (by marking them and pressing `Remove selected`), go to a new search to add other items (by pressing `Search new`), or to perform the purchase (by pressing `Buy`).

The described system is very simple and far from complete (e.g. no payment processing is considered), but it serves our purposes of demonstrating how this interaction can be specified and implemented well.

4. Interaction Design in Hera

In this section we design the example application using Hera and we point out how we can extend the Hera models and architecture to facilitate interaction design. Since the desired application structure is captured in AM, we focus on the AM specification.

4.1 Conceptual Model of the Poster Shop

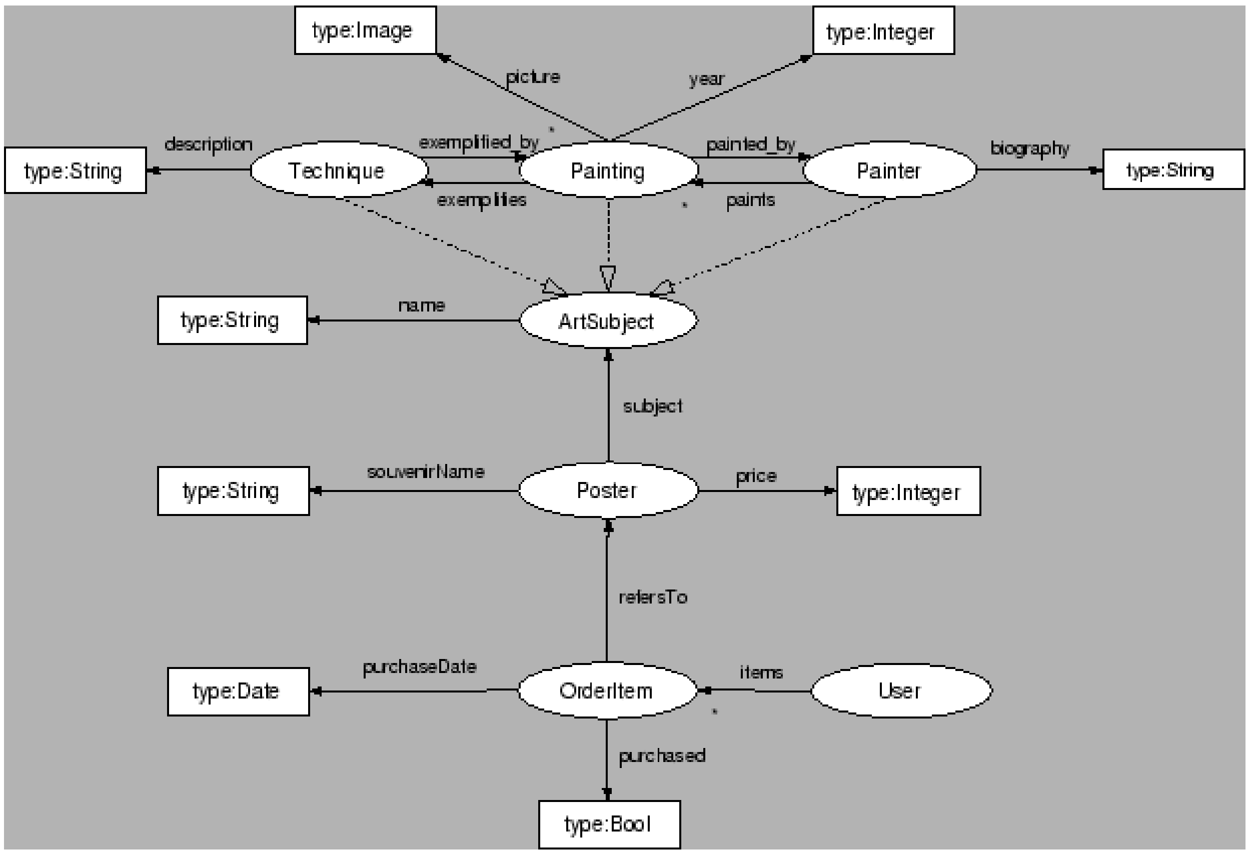

Figure 6: The CM of the poster shop

The CM of our example (see figure 6) contains the main concepts `Poster`, `ArtSubject` (covers paintings, techniques, and painters), and `OrderItem` (represents an item in the shopping cart). The `purchased` attribute of `OrderItem` determines wether the item was purchased or it is only in the shopping cart.

4.2 Application Model of the Poster Shop

The AM in Figure 7 outlines the envisioned application structure by means of the slices:

- `Poster.List` contains a list of (found) posters, where the user can choose the posters he wants to put into the shopping cart; pressing the `checkCart` button opens the cart report (`OrderItem.CartReport`); pressing the `buy` button performs the purchase and displays the purchase report represented by the slice `OrderItem.PurchaseReport`. The `Poster.List` slice corresponds to the `Found posters` page from Figure 5.

- `OrderItem.CartReport` contains a list of items in the shopping cart; the buttons there allow removing of items from the cart, displaying related items to these in the cart (posters with the same subjects), and performing the purchase. The `OrderItem.CartReport` slice corresponds to the `Shopping cart content report` page from Figure 5.
- `OrderItem.PurchaseReport` contains a list of purchased posters; the button `Search` opens the initial search page. The `OrderItem.PurchaseReport` slice corresponds to the `Purchase report` page from Figure 5.
- `Poster.ListItem` (nested in `Poster.List`) is an item in the list of found posters.
- `OrderItem.ListItem` is an item in the list of items (in a cart or purchased).

For the interaction we have extended the original definition of AM specification in two dimensions:

- The slices in AM can contain new elements that were not used in AM specifications before (e.g. button). They capture events caused by the user, or serve for data entering and output. The specification of these elements represents a *structural* extension of AM specification.
- Naturally, data and events provided by the user must be processed on-line. In addition, the data content of slices may depend on previously collected data. All this processing must be specified in some way. Therefore, it is clear that we need also a *functional* extension of slices that goes beyond simple navigation specification. This could include the manipulation of system/session state data, for instance the content of a shopping cart.

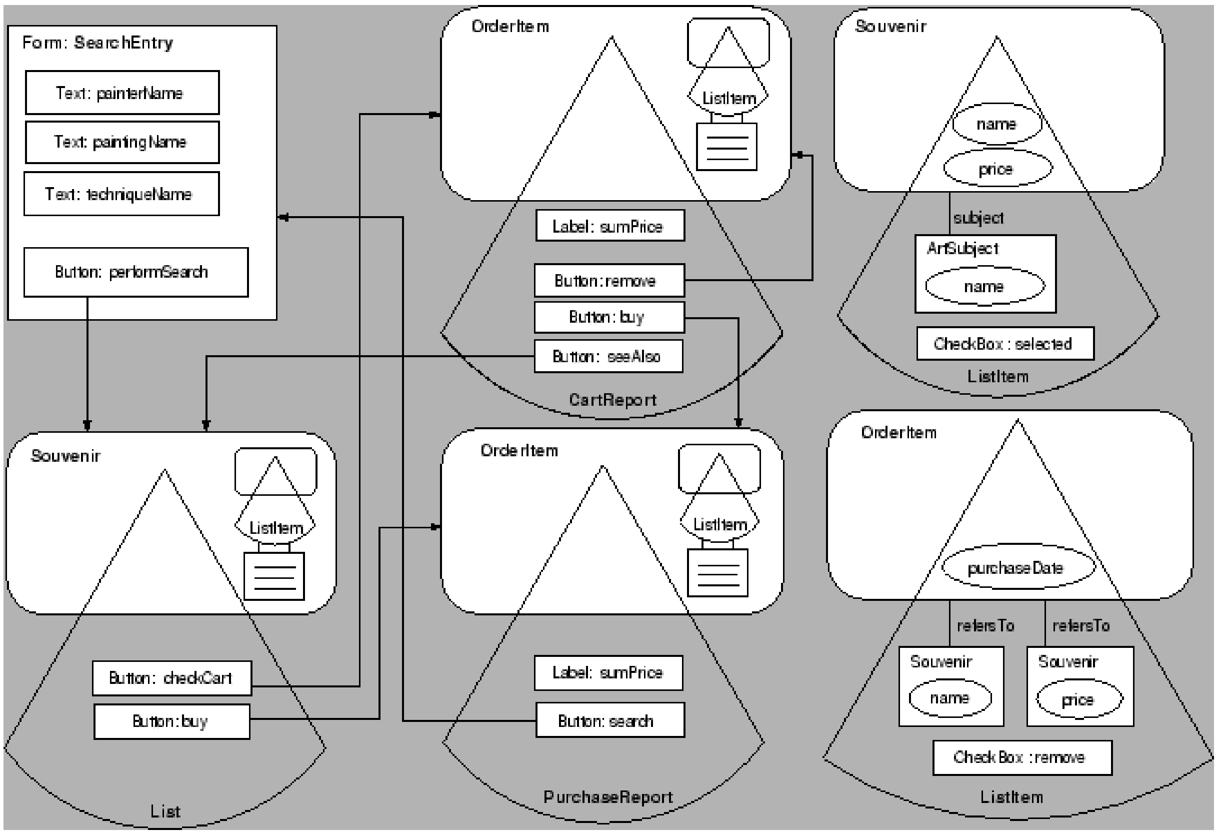

Figure 7: The AM of the poster shop

4.2.1 Structural Extension

As illustrated in Figure 7 we add to the AM definition new elements, called controls:

- **Button**. An example is the `Search` button in the `SearchEntry` form. It is obvious, that the button should be associated with operations to perform (as we explain further) when the button is pressed. The `performSearch` button should together with the `Poster.List` slice provide the list of matching souvenirs.
- **Checkbox**. An example is the `selected` checkbox in `Poster.ListItem`, where the user determines which items are put into the shopping cart. The checkbox control has assigned to it a boolean value that can be changed by the user.

- **Text entry field**. This field is needed for example for entering search criteria in the `SearchEntry` form; the value of a text entry field control is a string and can be changed by the user and read by the application.
- **Label**. Unlike the regular data slice attribute, a label is calculated from values of other attributes or controls. An example of the label is `SumPrice` in the `OrderItem.CartReport` slice. Labels are associated with assignment operations (e.g. an RQL query returning a single value).

4.2.2 Functionality Extension

Before we show how to specify the behavior of controls we need to realize the following:

- Due to user interaction that may influence the data content of slices we need to select the data content dynamically just before a slice is reached via navigation. The link, or link anchoring control should provide the condition selecting the data instances for the target slice. We call this **dynamic slice instantiation**.
- Thus generated hypermedia presentations are not static anymore, the system should maintain its **state information**: in our example the content of the virtual shopping cart. The system should be able to read and update this type of information.

Obviously, the structural diagram in Figure 7 does not describe the functionality related to controls, or the functionality related to data retrieval/update. For the sake of simplicity we show the functional specification only for the `Poster.List` slice. Let us assume that the `checkCart` button is pressed by the user.

- **The users's shopping cart** is updated according to the currently selected and unselected items in the souvenirs list:
  - **Selected posters are added to the shopping cart**. New instances of `OrderItem` are created. In RQL it can be:

    ```
    INSERT INTO OrderItem (refersTo, purchaseDate, purchased)
       FROM   X:ListItem
       VALUES {X}.root, null, 'false'
       WHERE  X.selected='true'
    ```

    The nested slice `ListItem` is referenced here in a similar way as a concept, and its attributes are referenced as concept attributes (`selected`). The `root` expression acting as a slice ``property'' returns the slice root concept (e.g. `Poster` for `Poster.ListItem`).
  - **Unselected souvenirs are removed from the shopping cart**. All `OrderItem` instances corresponding to the unselected posters with (the `purchased` property set to `false` means that the items are in the basket and are not purchased yet) are deleted. Using RQL we can express it:

    ```
    DELETE   X
       FROM  {X:OrderItem}refersTo{Y:Poster},
             {Z:ListItem}root{Y}
       WHERE Z.Selected='no', X.purchased='false'
    ```

- **The `OrderItem.CartReport` slice is instantiated and displayed**. The condition determining instances of the `OrderItem.ListItem` should be specified: all instances with the `purchased` value equal to `false`.

  ```
  REF( purchased='false' )
  ```

  Taking into account the structure of `OrderItem.CartReport` the system will execute this constructed RQL query:

  ```
  SELECT X.purchaseDate, Y.name, Y.price
  FROM   {X:OrderItem}refersTo{Y:Poster}
  WHERE  X.purchased='false'
  ```

  The structure of the `SELECT` clause is based on the attributes of the target slice `OrderItem.CartReport` with nested `OrderItem.ListItem`.

Since the principle is similar for the `buy` button, we do not show the specification for it. The specification can be serialized into an RDFS file. The sample pattern of such a file is:

```
<rdfs:Class rdf:id="Slice.Poster.List">
<rdfs:SubClassOf rdf:ID="#Slice">
...
<rdfs:Class rdf:ID="checkCart">
  <rdfs:SubClassOf rdf:resource="#Button"/>
  <rdfs:Class rdf:id="checkCart-processing">
    <rdfs:SubClassOf rdf:resource="#Processing"/>
    <rdfs:Class rdf:id="InsertItem">
      <rdfs:SubClassOf rdf:resource="#Operation"/>
        <AMS:OpBody>
          INSERT INTO OrderItem (refersTo,purchaseDate, purchased)
          FROM    X:Slice.OrderItem.ListItem
          VALUES  {X}.root, null, 'false'
          WHERE   X.selected='true'
        </AMS:OpBody>
      </rdfs:Class>
    </rdfs:Class>
  ...
  </rdfs:Class>
</rdfs:Class>
```

The `X` argument is bound to the `OrderItem.ListItem` slice, and the RQL command using it creates an instance of the `OrderItem` concept. The `owner` here is not the name of an slice attribute, but refers to the owner concept of the slice `OrderItem`. The AMS is a namespace of the AM schema, which defines among others also the `Slice`, `Button`, `Processing`, and `Operation` concepts.

5. Architecture of interactive WIS

The architecture of WIS should be refined to allow this kind of interaction. Mainly, there is a need of an execution engine that would perform dynamic slice instantiation based on query processing and data retrieval, and data updates. The engine should provide:

- Reference operations: the system processes the operation and returns a slice instance(s) in the form of the element (e.g. HTML page) to be displayed next, where it needs to be performed:
  - construction of the RQL query from the target slice structure (`SELECT` and `FROM` clauses of an RQL query from the slice structure and the construction of the `WHERE` clause from the condition) and from the selecting condition that is the part of the reference operation,
  - execution of the query, and
  - translation of the raw data into slice instance(s) and then into the presentation unit (in whatever format, e.g. HTML).
- Data manipulation operations, and other RQL queries: the system evaluates all references in queries and executes them.
- External calls (e.g. for on-line payment, but also same other more complex functions as complete shopping carts), typically static calls of web services (using SOAP and WSDL) and possibly calls of dynamically discovered (loosely coupling) web services using UDDI, and perhaps DAML-S, etc.

5.1 Implementation Issues

To make the system versatile, the engine performing dynamic generation of presentation pages (based on dynamic slice instantiation) and processing user events/data would include all engines from figure 4. The prototype system we developed runs as a servlet under a host web server (Apache Tomcat). This servlet processes the `Get` (HTTP client asks for another page that should be provided) and the `Post` (the client sends an event to the server and values of controls are read) HTTP messages.

As a response to the `Get` message the system:

- determines the next slice from AM that will be rendered as the next page,
- performs a data query constructed from the selecting condition associated with the reference operation and from the structure of the target slice,
- retrieves data (creates a CM instance) and creates an instance(s) of the target slice (an AM instance), and
- transforms the AM instance into the presentation page in an appropriate format (e.g. HTML)

As a response to the `Post` message the system will collect the data provided by the user interacting with concrete controls. The system reads data values of controls that are passed to the `Post` message as arguments. Instances of controls are bound with concrete slice instances during page generation via forms and nested hidden arguments (`<form/>` in XForms and HTML).

6. Conclusion

The ideas proposed here point to extension of our models, schemas, and architecture regarding the design of interactive WIS. We have demonstrated the need of interaction in typical e-commerce applications. With this paper we have established the direction of the research and for now omitted exhaustive schema specifications and complete sets of controls and corresponding mechanisms. We have indicated how these ideas can be implemented on the basis of our prototype.

Another aspect that is a subject of our intensive research and is not sufficiently covered yet, is automation of the design process of such interactive presentations. There are several possible ways that we investigate, for instance re-use CM and AM patterns, or semi-automated generation of AM from CM and formalized goals/tasks of the system.

References

1. Brickley, D., Guha, R.V.: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Working Draft (2002
2. Ceri, S., Fraternali, P., Matera, M.: Conceptual Modeling of Data-Intensive Web Applications. IEEE Internet Computing, vol. 6, number 4, pages 20-30 (2002)
3. De Bra, P., Aerts, A., and Houben, G.J., Wu, H.: Making General-Purpose Adaptive Hypermedia Work. In Proc. WebNet World Conference on the WWW and Internet, AACE (2000)
4. De Bra, P., Houben, G.J., Wu, H.: AHAM: A Dexter-based Reference Model for Adaprtive Hypermedia. In Proc. The Tenth ACM Conference on Hypertext and Hypermedia, ACM Press (1999)
5. Frasincar, F., Houben, G.J., Vdovjak, R.: Specification Framework for Engineering Adaptive Web Applications. In Proc. The Eleventh International World Wide Web Conference, Web Engineering Track, ACM Press (2002)
6. Gervais, M.P.: Towards an MDA-Oriented Methodology. In Proc. 26th Annual International Computer Software and Applications Conference, IEEE Computer Society (2002)
7. Hera web page and software prototype. Available on the URL `http://wwwis.win.tue.nl/~hera`
8. Isakowitz, T., Stohr, E., Balasubramanian, P.: RMM: A Methodology for Structured Hypermedia Design. Communications of the ACM, volume 38, number 8, pages 34-44 (1995)
9. Karvounarakis, G., Alexaki, S. Christophides, V., Plexousakis, D., Scholl M.: RQL: A Declarative Query Language for RDF. In Proc. The Eleventh International World Wide Web Conference, ACM Press (2002)
10. Kay, M.: XSL Transformations (XSLT) Version 2.0. W3C Working Draft (2002)
11. Koch, N., Kraus, A., Hennicker, R.: The Authoring Process of the UML-based Web Engineering Approach. In Proc. First International Workshop on Web-Oriented Software Technology (2001)
12. Lassila, O., Swick, R.R.: Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation (1999)
13. Schwabe, D., Rossi, G., Barbosa, S.D.J.: Systematic Hypermedia Application Design with OOHDM. In Proc. The Seventh ACM Conference on Hypertext (1996)