Hera-S - Web Design Using Sesame

Kees van der Sluijs[1], Geert-Jan Houben[1,2], Jeen Broekstra[1,3], Sven Casteleyn[2]

[1]Technische Universiteit Eindhoven
PO Box 513,
5600 MB Eindhoven,
The Netherlands
+31 40 247 2733
{k.a.m.sluijs, g.j.houben, j.broekstra}@tue.nl

[2]Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels, Belgium
+32 2 629 33 08
{Sven.Casteleyn, Geert-Jan.Houben}@vub.ac.be

[3]Aduna
Prinses Julianaplein 14b,
3817 CS Amersfoort,
The Netherlands
+31 33 465 9987
jeen@aduna.biz

ABSTRACT

Web application design methods traditionally aim to reduce complexity in implementing Web applications. However, these methods struggle with providing the necessary dynamics and flexibility to keep up with the increasing users' demand for personalization and feedback mechanisms. We present Hera-S, based on Sesame and the SeRQL query language, that does provide the necessary flexibility. Hera-S allows designers the plain use of the Semantic Web languages RDFS and OWL for designing the domain model and the context data model, thus enabling re-use of existing data models and opening up the RDF instance data to queries and updates via the Sesame RDF-framework. Furthermore, Hera-S provides an increased flexibility by integrating server-side scripting availabilities (e.g. for the integration of web services) and capabilities for adding specific code constructs as often used in manually crafted Web applications (e.g. JavaScript). In this way it is able to seamlessly integrate existing solutions, without losing the complexity-reduction features necessary for rapid, easy and error-free Web application design and deployment.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *User-centered design*; H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia – *Architectures, Navigation*.

General Terms

Design, Reliability.

Keywords

Hera-S, Sesame, Web information system, Semantic Web, Application Model, navigation, dynamics, RDF-framework, personalization, adaptation, user feedback.

1. INTRODUCTION

The dynamics and flexibility required in modern Web information systems (WIS) demand a lot from WIS design methods. In their model-based approach to limit and control the complexity, their navigation-based nature often has difficulties accommodating the requirements that evolve from an increasing demand for personalization, feedback, and interaction mechanisms. There are limitations to the (modeling) constructs that can be used in the models. It is difficult to integrate external service elements (positive example in [2]) and there is often no possibility to add specific client-side functionality (positive example in [5]) that is available in most Web browsers. This difficulty in connecting to third-party software (e.g. for business logic) is also reflected in the difficulties around extending the application by integrating different data sources. The inclusion of additional (background) knowledge is no sinecure. In this paper we describe Hera-S that meets a number of these requirements by offering additional constructs and that allows the application to connect to the content and context data through the Sesame RDF framework (e.g. [3] and [11]). This RDF-based solution, combining Hera navigation design and Sesame data processing by associating SeRQL queries to all navigation elements, allows to effectively apply existing Semantic Web technology and a range of solutions that is becoming available. We can thus include background knowledge (e.g. ontologies) and connect to services through the RDF-based specifications. We can also use the facilities in Sesame for specific adaptation to the data processing and to provide more extensive interaction processing. In the next section of this paper we explain the ratio behind the combined approach. Sections 3 and 4 discuss the models that are used in the design, both for the data and its processing in an application. Section 5 then addresses the implementation, while section 6 considers some scenarios that are thus supported.

2. METHOD

The purpose of Hera-S is to support the design of navigation-based Web structures over semantically structured data. Figure 1 shows an overview of the Hera-S method.

Hera-S takes as a starting point a domain model (DM) that describes the structure of the content. Based on this DM, the designer creates an application model (AM) that describes a hypermedia-based navigation structure over the content for the sake of delivering and presenting the content to the user.

Hera-S allows dynamic personalization and adaptation of the content. For this purpose, context data is maintained (under control of the application) in a so-called context model (CM). This context data typically is updated based on the (inter)actions of the user as well as on external information (more on this in section 3).

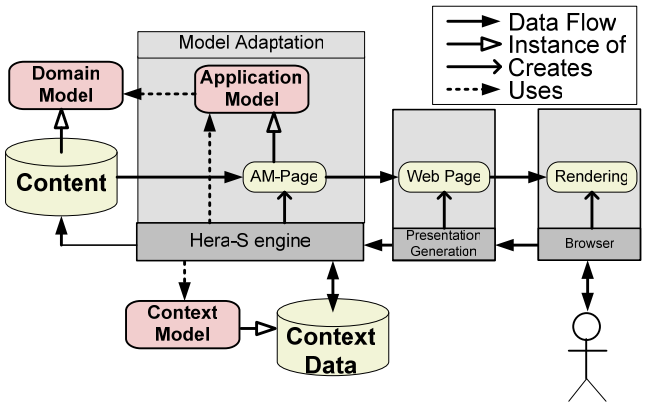

Figure 1: Hera-S Overview

Instantiating the AM with concrete content subsequently results in AM (instance) pages (AMP). These AMPs contain the navigation primitives (based on underlying semantic relations from the DM) that can be used by the user to navigate to other AMPs and thus to semantically "move" to a different part of the content. An AMP itself is not yet directly suitable for a browser, but can be transformed into a suitable presentation by a presentation generator. In previous publications, we have demonstrated that both proprietary [13] and external engines (e.g. AMACONT [7]) can be used for this task. For this paper and for Hera-S this presentation generation phase is not relevant, and as we will see in Section 6 we can apply a direct AMP-to-HTML conversion (e.g. using XSLT) and use a CSS file for presentation formatting. We already remark that Hera-S supports adding to the AMP browser-specific data which depends on the presentation target.

AMP creation is pull-based, meaning that a new AMP is only constructed by the Hera-S engine at request (in contrast to constructing the whole instantiation of the AM at once). By navigating (link-following) and forms submission the user triggers the Hera-S's feedback mechanism, which results in internal updating/querying of the website navigation or context data and creation of a new AMP.

Next to this, the architecture of Hera-S allows to offer an additional service for designers, namely an API for model adaptation. This API can be used by external applications to interfere with the adaptation (e.g. to provide aspect-oriented adaptation support or to include external navigation templates [1]).

3. DOMAIN AND CONTEXT DATA MODELS

Hera-S expects Domain Models in RDFS or OWL format (without restrictions). As we use the Sesame RDF Framework [3] as our back-end repository, this data can be exploited and reasoned upon. Accessing the content data in Hera-S will be done via customizable SeRQL [4] queries, so that the full potential of the SeRQL query language can be used for AMP creation. Furthermore, content that is under the designer's control can also be updated via Sesame's access API.

By supporting unrestricted RDFS or OWL DM's, Hera-S is particularly suited to (re-)use existing domain ontologies. Moreover, many existing data sources that are not yet available in RDFS or OWL format can be used via Semantic Web wrapping techniques (e.g. [16] or [14]). In the latter case Sesame can be used as a mediator between such a data source and Hera-S.

The CM is modeled and implemented in a similar way as the DM. The main difference between the two is that the context data is assumed to always be under direct control of Hera-S, while the content often is not. Furthermore, the content typically contains the information that in the end is to be shown to the user, while the context data typically contains information used (internally) for personalization and adaptation of content delivery. This distinction might not always be strict, but as it is only a conceptual distinction in Hera-S, the designer may separate content and context in whatever way he desires.

As the CM is also implemented with a Sesame repository, it also provides the means for other processes to use (some) and update (some of) this information. The context data could for instance be manipulated by business logic software for the sake of adaptation, or by external user profiling software.

Another great advantage of using Sesame is the possibility to combine several data sources (both content and context data) at the same time (we will give an example of this in section 6). In this way, designers can couple additional data sources to the already existing ones and can thus easily extend the domain content. This also offers possibilities to exploit additional knowledge when performing a search. Currently, we are experiencing the exploitation of the WordNet ontology [10], [12]), time ontologies (e.g. [8]) and geographic ontologies (e.g. [15] and [6]). In this way, a keyword-search can be extended with synonyms extracted from the WordNet ontology, or a search for a city can be extended with surrounding cities from a geographic ontology. In the example scenarios in section 6 we will shortly indicate where we could use this additional information.

Even though the designer is free to choose any kind of context data, we in general discern three types: session data, user data and global data.

Session data: Session data is data relevant to a certain session of a certain user. An example of such data is the current application context, such as the device that is used to access the Web application or the units browsed in the current session.

User data: Data relevant to a certain user over zero (i.e. initial user data) or more sessions. User (profile) data can be used to personalize (even at the beginning of a new session). Note that for maintaining this user data over time the application needs some authentication mechanism.

Global data: Usage data relevant to all users over all sessions. Global data typically consists of aggregated information that gives information about groups of people. Examples include "most visited unit" or "people that liked item *x*, also browsed item *y*".

4. APPLICATION MODEL

In the Application Model (AM), the navigational behavior of the Web application is specified. The AM enables designers to specify which objects are shown to the user and what Web pages the user can navigate to. Furthermore, the AM allows a dynamic specification of the data, such that the data can be personalized to a user and adapted for a specified context.

The AM is specified by means of *Navigational Units* (shorthand: *unit*) and *Navigational Relationships* (shorthand: *relationship*) between those units. Elements instantiating the units, and relationships, can be defined by queries that refer to the content and context data as explained in section 3. Let's now discuss in detail the navigational units and relationships, and their sub-elements, which together make up the AM, before we illustrate the implementation in the next section.

Navigational unit: A navigational unit is an application model element that represents some semantic content which together forms a logical unit to the users, and will thus be presented together. Every unit in an AM is uniquely identified. The actual content of a unit is specified by its *access query*, and by (the access query of) its sub-elements. The access query is expressed in a SeRQL query that may search for information in the domain and context. Note that the information that is available to a unit is also accessible by all its sub-elements. Navigational units may contain sub-elements and also other navigational units. We distinguish two types of sub-elements, namely simple and complex ones. As simple ones we discern attributes and constants:

- **Attribute**: Attributes are the result of a query to the data sources which returns exactly one result. Such a query may involve more than one data source and may also contain variables that are determined by the access query of the containing unit. As the data sources consist of RDF data, attributes are typically the literal values of some property. We will regard literal values by default as strings, unless they are explicitly typed as an XML data type. References to Web objects are assumed to have the xsd:anyURI type. For more specific processing of multimedia objects such as pictures or applets, we support a proprietary mimeType property (currently "text/html", "image/*" and "application/x-java-applet").
- **Constant**: A constant is an element that contains fixed data to be shown to the user, i.e. data which is not queried from the content or context data sources. Examples of constants are header and footer text, a welcome message, etc.

Besides simple sub-elements we also regard a more complex structure, namely sets.

- **Set:** A set is a collection of zero or more elements Sets provide mechanisms to specify (display) operations both on the whole set and on all the individuals of the set. The content of a set is defined by an access query. We also offer a special set construct, namely the (guided) tour.
    - **Tour**: A sequence of navigational objects which are shown one at a time and can only be navigated in the order of the sequence.

Besides elements targeted at the display of information, we also want the user to be able to navigate through the data and interact with (provide feedback to) the system. Specifically for navigation we introduce the sub-element anchor and for feedback-interaction we introduce the sub-element form.

- **Anchor**: An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. They can be used for the navigation between units, or between a unit and an external source. Furthermore, anchors can be used as a destination part of a Web page for incoming links. Anchors can be attached to attributes, sets and constants. Anchors are closely related to navigation relationships, which we will explain later on.
- **Form**: Forms provide a means to request manual input from a user (besides clicking link anchors), but also provide an output mechanism (e.g. a pre-filled form), and a way to (visibly or invisibly) transport information between units. A form contains a number of form elements (e.g. string input, selection element, hidden input elements, etc), but may also contain attributes and sets. Forms can be made dynamic by using sets to determine the number and type of form elements (as will be further demonstrated in section 6.1).

Sometimes, a Web application might need more expressive power or some specific constructs for certain browsers. Furthermore, within an AM one might want to utilize the functionality of Web services or other server-side functionality. Therefore, we have two additional sub-elements called code objects and script elements.

- **Code Object**: Object containing some 'code' that will be executed by the client browser. Code objects are not processed by the engine executing the AM, and are only included in the output. Code objects are similar to constants, but differ in that they can (but don't have to) be "attached" to other elements (typically to react on events within that elements). Typical example languages for code objects are HTML, JavaScript, CSS, SMIL, etc.
- **Script element**: Navigational Units may contain inline script elements. A script element contains scripting code that is executed server-side before construction of the unit, e.g. by calling a Web service. The execution possibly returns output in the form of any of the previously described elements, which will subsequently be integrated in the unit. A script element can be defined in a scripting language like PHP, JSP, etc, depending on which of these languages is supported by the server that runs Hera-S.

The standard navigational behavior of a navigational unit is that, when a link is followed, the current unit (i.e. the one containing the source anchor) will be replaced by the link's target. However, we also allow links to replace other units' content by the link's target, similar to the HTML frame-construction. For this end the links should specify which unit should be replaced.

While the units define the content, for the navigation in Hera-S we use the concept of navigational relationships between units.

Navigational Relationship: A navigational relationship is a representation for the directed relationship between a source unit and a target unit (except for the session start, which is external). This relationship is the basis for navigation in the hypermedia application (i.e. through hyperlinks and forms). The relationship is constituted of a query that refers to the target unit, and also specifies adaptation of that unit by means of queries to domain and context data. This query consists of three (possibly empty) parts: 1) a query for pre-updating data in the data sources based on input from the source unit, 2) a query that retrieves data from the data sources for (composing) the target unit, 3) and finally a query that post-updates data in the data sources based on processing of the target unit. We note that the two update parts are motivated by the need to separate updates that take place prior to the target unit composition from those that take place afterwards.

One special navigational relationship is the 'contains' relationship to define the elements contained in a navigational unit.

We have now defined the main concepts of the Hera-S models. In the next sections we will treat the implementation of Hera-S and example scenarios in which we sketch how a particular Web application can be constructed using Hera-S.

5. IMPLEMENTATION

In this section, we will illustrate the implementation of Hera-S, and show how the Sesame framework [3] and the SeRQL [4] querying facilities are used to realize the system.

5.1 Hera-S Implementation Architecture

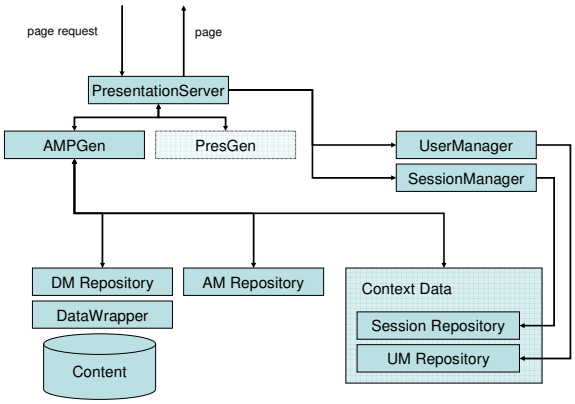

Figure 2: Implementation Architecture

In Figure 2 we see the main components that make up the Hera-S implementation architecture. The domain model (DM), application model (AM) and all context data are realized as Sesame *repositories*, exploiting Sesame's capability that enables storage, reasoning and querying of RDF and OWL data.

The content is interfaced to the rest of the system through the DM repository. A major advantage of this approach is that integrated querying of both schema and instance data becomes possible. To enable this, the content has to be represented as RDF statements. For non-RDF content repositories this can be achieved in various ways. The simplest and most straightforward case is an offline translation of the data to RDF and simply storing that RDF in a Sesame repository. However, this approach has a drawback by duplicating data which means that updates to the data need to happen in two places. An alternative way of realizing the link between DM and data is by creating a wrapper component around the actual data source that does online back-and-forth translation. The Sesame architecture caters for this scenario by having a storage abstraction layer called the SAIL API [3]. A simple wrapper component around virtually any data source can be realized as a SAIL implementation and then effortlessly be integrated into the rest of the Sesame framework and thus into our Hera-S environment.

The entire system is an event-driven architecture. When a request for a certain page comes in at the Presentation Server component, the request is translated to a request for an AMP. At the same time, the UserManager and SessionManager components are informed of the request. These two manager components can then take appropriate actions in updating the context data repositories (specifically, the UM (user model) Repository, and the Session Repository). Independently from this, the so-called AMPGen component retrieves the requested part of the AM that contains the conceptual specification which is the basis for the next AMP. It then starts the AMP creation process by following that specification. The AM specification consists of constructs and their queries that specify the information from the DM and CM that is needed to correctly instantiate the AMP.

The actual AMP is internally implemented as a volatile (in-memory) Sesame repository, which means that all transformation operations on it can simply be carried out as RDF queries and graph manipulations using the SeRQL query language. When the AMP has been fully constructed, it is sent back to the server. The implementation architecture allows for a presentation generator component, which, as already explained in section 2, can be any third-party component. This presentation generation component can then transform the AMP into an actual page (in terms that a thin client such as a Web browser can understand, e.g. XHTML). The result is then finally sent back to the client (as the response).

5.2 RDF Querying in SeRQL

SeRQLis a declarative query language for RDF, implemented in the Sesame framework, which allows flexible manipulation of RDF graphs. Queries are defined by a *graph template pattern*, a set of Boolean constraints and a set of filter conditions.

An example SeRQL query is:

```
SELECT movie, title
FROM   {X} rdf:type {imdb:Movie};
           imdb:title {title}
```

This query retrieves the movie identifier and its title attribute from a collection of movie descriptions (see also section 6, Example scenarios). Notice that in the FROM clause a graph pattern template is specified – each statement is represented by a pattern of the form '{Subject} Predicate {Object}', that is, each graph node is surrounded by curly brackets. Path branching is indicated by a semicolon: in the above query, the semicolon indicates that both the predicate 'rdf:type' and 'imdb:title' emanate from the subject variable X..

SeRQL supports two basic projection types: *variable binding*, realized by SELECT-queries as shown above, and *triple pattern generation,* realized by CONSTRUCT-queries. Especially the latter is useful for data transformations, since it allows the transformation of one graph into a different graph. A CONSTRUCT clause is used in place of the SELECT clause and rather than a list of variables, it too specifies a graph pattern template, which in this case prescribes the graph structure of the query result.

In the Hera-S system, and mostly in the access queries, SeRQL query expressions are extensively used to define mappings and filters between the different data sources source data and the eventual AMP: after all, since all this data is expressed as RDF graphs, an RDF query/transformation language is a natural choice as a mapping tool.

In the next section, we will outline some example scenarios and show example SeRQL queries that are used to map content and context data to the application model, i.e. from DM/CM to AM.

6. Example scenarios

In our example we use the movie domain as a case study. Our goal is to create a personalized Web application, similar to the well-known International Movie Database (IMDB, [9]) application. Suppose we have a (simplified) domain model (DM1) representing information about movies and their crew (e.g. actors, directors), and a context model (CM1) representing information about the users' viewing history (e.g. the number of times users viewed a particular movie, and an aggregation of all the views of every movie by all users). This is depicted in the UML diagram in Figure 3.

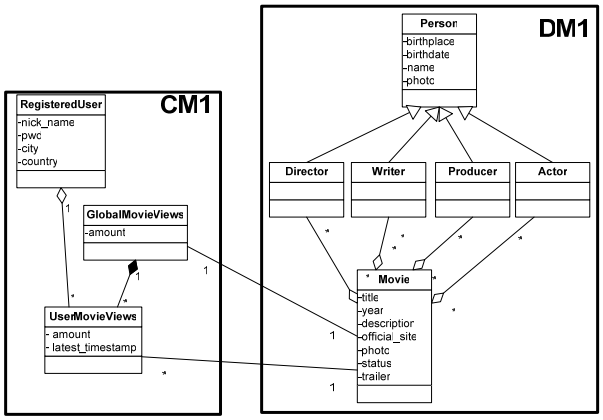

Figure 3: IMDB domain and context data model

In our case study we will consider a simple part of the application that allows navigation over movies and persons. In section 6.1 we give an example of personalization in this application and in section 6.2 we give an example of how an existing domain can be extended by adding cinema information to the movie application.

6.1 Personalization

In the first example scenario we will look particularly at personalization. Therefore, the application will start with a login screen. After authentication, the user arrives at a starting page that contains a menu (which will remain visible throughout the whole browsing session) and a welcome screen. Within the menu the user has the option to logout, go to a search page, go to a favorites page and see a history of the last 10 movies that were browsed.

The 'search' page enables users to search over the movies and persons in the database. There are two kinds of search. Simple text search will find all persons and movies that contain the search string in one of their attributes. More advanced is the attribute search, which allows searching for persons and movies by querying specific attributes. Note that the types of search are only restricted by the expressive power of the SeRQL language: here we opt to use simple types of search.

The 'favorites' page shows a list of the top 10 most viewed movies for the current user. Furthermore, it will also contain a list of the top 10 movies viewed by all users. These two lists are made "expandable" and "collapsible" via the user interface (similar to existing well-known hierarchical (directory) structures). Similarly, the 'history' page shows the ten last movies and persons that have been visited.

Evidently, movies and persons can be browsed the 'regular' way. This means that every movie has a separate page that shows the user a collection of its attributes (title, year, description, etc). It also shows a list of people (by their name) that are involved in the movie, and are grouped based on their function (actor, director, etc). Every person name has a hyperlink to his/her personal page. On this page a number of his/her attributes (e.g. DateOfBirth, Name) are shown. Furthermore, for every person a set of movie names (movies in which they were involved) is shown. This set is grouped by function (director, actor, etc). Every movie name again links to the corresponding movie page.

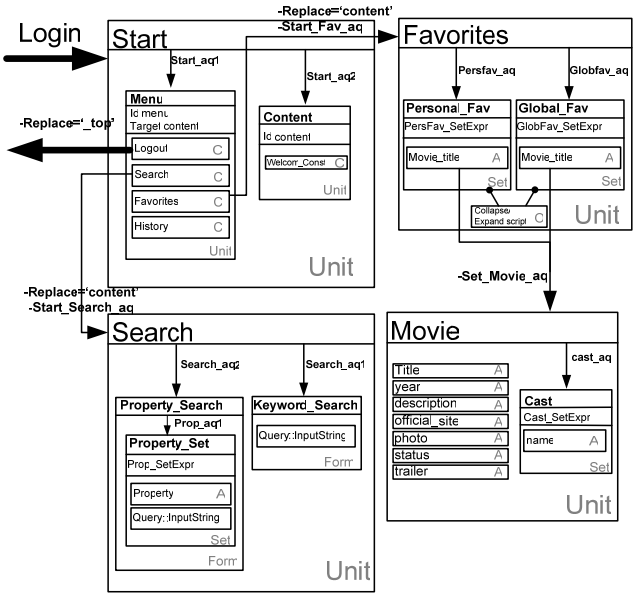

Figure 4: Partial AM for personalization example

Figure 4 is a partial visualization of the AM for our application. Because of space considerations it only shows the most interesting units, omitting units like *Login, History, SearchResults* and *Person.*

The *Start* unit succeeds a login process. Therefore, it has login information at its disposal that can consequently be used by its sub-elements. The *Start* unit has two sub-units: *Menu* and *Content*. *Start_aq1* is the access query for *Menu* and is used to personalize the anchors that are attached to the *Favorites* and *History* menu items, such that navigational relationships will transport the required information for the relevant units. *Start_aq2* in this case may be left empty, as the *Content* unit only contains one constant without an anchor (i.e. a welcome message).

The anchors of the elements in the *Menu* unit also have, besides a navigational relationship, a "Replace" specification. This specification indicates that not the current unit (here: *Menu*) will be replaced, but the unit with the name specified in "Replace" (in this example *Content*).

In the *Search* unit, a simple *Keyword_Search* form is shown and also a more specific *Property_Search* form. The *Property_Search* in our example is a dynamic form which is created based on an input query. The *Property_Set* access query queries for all property names for movies and persons, as defined in the Domain Model. As a result, for every property its name is shown, together with a (string) input element. Using such a (generic) input query as a basis of the property-search form yields the advantage that

whenever the Domain Model is adapted, the *Property_Search* form is automatically adjusted.

The *Favorites* unit contains two sets. One set shows the top 10 movies that are most often viewed by the current user. The other set shows the top 10 movies that are most often viewed by *all* users. Both sets get the same code object attached. This code object takes care of collapsing or expanding the sets. If we target HTML, we could (in our case) for example insert JavaScript code that sets the display-style to none for collapse, and unsets this style for expand.

Let's now go in more detail into the unit representing the top 10 movies of the current user. Figure 5 contains a snippet of the underlying code of the *Favorites* unit, and in particular the personal favorites. It shows the SeRQL (access) query for selecting the names of the 10 most viewed movies by the current user. Furthermore, these names are ordered on (amount of) views and limited to the top-10 at most.

```
<NavigationalUnit rdf:ID="Favorites">
 <SubElement>
  <Set rdf:ID="Personal_Favorites">
   <Set_AccesQuery rdf:datatype="xsd:string">
   SELECT DISTINCT MovieTitle
   FROM {} rdf:type {imdb:Movie};
           imdb:title {MovieTitle};
           um:userMovieViews {} um:ofUser {User};
                               um:amount {Views}
           WHERE User = $$currentUser$$
           ORDER BY DESC(Views)
           LIMIT 10
   </Set_AccesQuery>
  <Setelement_Operator rdf:datatype="xsd:string">
   <Attach rdf:resource="Anchor_PersFav_Movie_aq">
     <Assign>
     $$movieX$$=MovieTitle
     </Assign>
   </Attach>
  </setelement_Operator>
 </Set>
 </SubElement>
 <SubElement>
  <Anchor rdf:ID="Anchor_PersFav_Movie_aq">
   <Anchor_Uses_NavigationalRelationship
              rdf:resource="#Fav_Movie_aq"/>
   </Anchor>
 </SubElement>
..............
</ NavigationalUnit>

<NavigationalRelationship rdf:ID="Fav_Movie_aq">
 <targetUnit rdf:resource="#Movie"/>
 <selectQuery rdf:datatype="xsd:string">
   {movie=$$movieX$$}
</selectQuery>
 <updateQueryPost rdf:datatype="xsd:string">
 {UPDATE {x} um:amount {views + 1}
  FROM {movie} um:amountOfViews {x} rdf:type
                                {um:UserMovieViews};
                           um:ofUser {user};
                                um:amount {views}
  WHERE movie=$$movieX$$ and user=$$currentUser$$
  },
  {
  UPDATE {x} um:amount {views + 1}
  FROM {movie} um:amountOfViews {x} rdf:type
                               {um:GlobalMovieViews};
                                um:amount {views}
  WHERE movie = $$movieX$$
  }
 </updateQueryPost>
</NavigationalRelationship>
```

Figure 5: AM code snippet for Favorites Unit

The code snippet also shows the definition of a navigational relationship. In the specification of these relationships, the queries that are executed when one of the movies in the *Favorites* page is selected are defined. In this case, the query specifies exactly which movie the *Movie* unit should show. As we already know which movie this should be, namely the movie that the user just selected, no query needs to be actually executed; instead this information just needs to be passed on, The update queries define an update of the amount of movie-views of the movie the user is navigating to, and likewise an update of the global movie-views of that movie. This update is only executed after the next unit is computed, so that the update does not interfere with the (current) visualization of that unit. Note that it is also possible to define pre-update queries (instead of post-update queries), so that the updates could influence the next unit.

Also note the use of variables in Figure 5. In the code snippet two variables are used, namely "$$movieX$$" and "$$currentUser$$". These variables have to be determined before the queries can be executed. The value of the $$currentUser$$ variable is resolved after the *Login* process succeeded. Subsequently, the *Start*-unit has this information to its disposal, and therefore also all of its subunits (i.e. through an access query, or through an additional query to the Context Data, that can also act as a temporary variable store). The value of the "$$movieX$$" variable is determined by attaching the navigational relationship as an anchor to all elements (via the *setelement_Operator*)of the *Personal_Fav* set, and substituting MovieTitle for $$movieX$$ by using the "Assign" construct.

After creation of an AMP a presentation generation process is applied before showing the data to the user. In our example, we applied a simple AMI-to-HTML converter and attached a CSS file to obtain the screenshot shown in Figure 6.

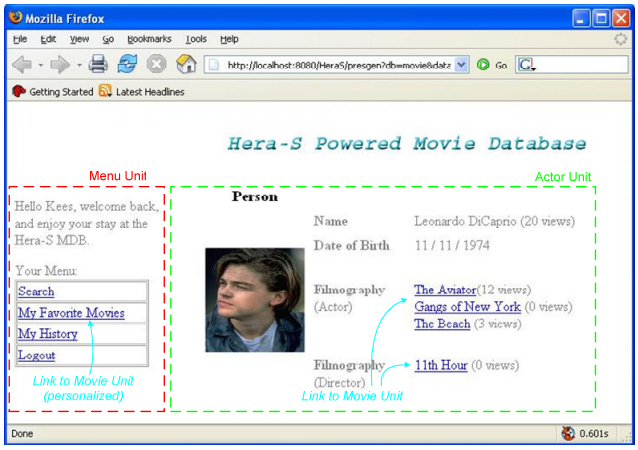

Figure 6: Hera-S Movie Database Screenshot

6.2 Domain extension

For our second scenario we consider an extension to an application (integrating for example a new source). We extend/unite the DM1 of Figure 3 with a new DM2 that maintains content about cinemas and their featured movies (note that DM1 and DM2 are modeled by two different domain ontologies). Furthermore, we store additional context data (CM2) that maintains a shopping cart and the items in this cart, related to the movie-showings a user wants to visit. The shopping cart is related to DM1 via the user data. These extensions of the models are represented in Figure 7. The advantage of using Sesame here is that this information can be easily stored in different data stores, without complicating the creation of a new AM. This is possible because in RDF one can easily cross-reference data between schemas.

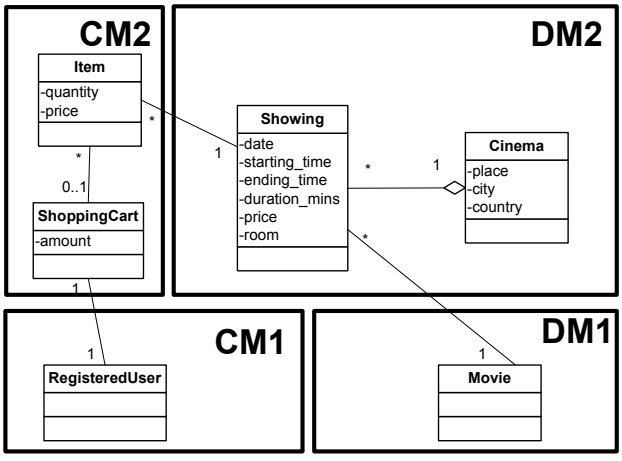

Figure 7: Additional content and context model

With this domain extension, let's now consider an extension of our application as described in Figure 4, in which we are also able to offer to the user information in which cinemas each movie is played. We can compute this information, because we refer (in 'showing') to a specific movie specified in DM1. The resulting query could then be (omitting sorting- or grouping-details):

```
SELECT DISTINCT Cinema
FROM {} rdf:type {cin:Cinema};
     cin:Showing {} imdb:Movie {} imdb:title
                                  {MovieTitle}
  WHERE MovieTitle = $$movieX$$
```

In the same way as extending the domain we can use additional context information. We maintain a shopping cart that is connected to a particular user. Furthermore the items that can be added to this shopping cart are related to particular movie showings. This information is sufficient to implement the shopping-cart functionality. At checkout the user can provide some payment details, which can then be updated in the CM. Third-party software, e.g. business logic software, can extract this data from the context data via Sesame, and actually process the payment and send the tickets.

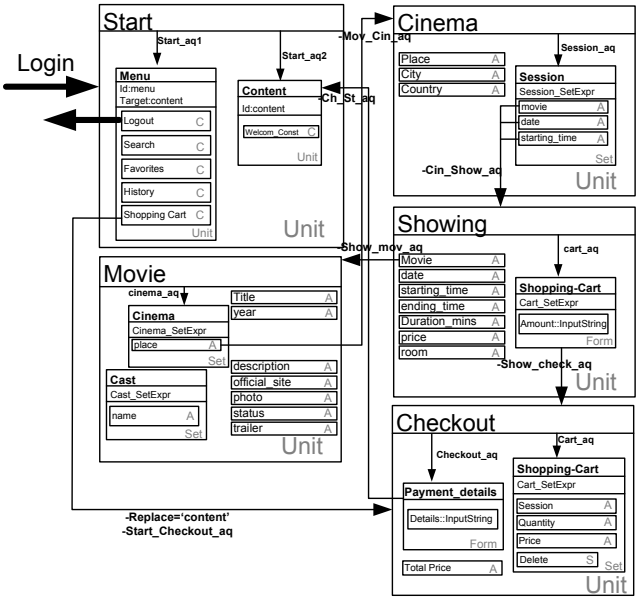

Figure 8: Partial AM for the domain extension example

Figure 8 contains the (partial) AM for this extended scenario. As it is an extension of the AM shown in Figure 4, we only show the relevant units that were changed, and the new units we added. Using Sesame, everything can be seamlessly integrated in the original AM. As the units have a similar structure as defined in Section 6.1, we will not treat them in depth again.

We can indicate one noticeable point, however. In the *Movie* unit, we now show all cinemas that show the specific movie. However, popular movies may be shown in cinemas all over the world, so probably the designer wants some mechanism to restrict this. By using additional information from an external geographical ontology, which we can (again) access via Sesame, we are able to use the location of the current user (via RegisteredUser.city as specified in CM1) to select only those cinemas (via Cinema.city as specified in DM2) in the neighborhood of the user.

7. CONCLUSION

In this paper we have discussed the design support for the complexity around dynamics and user-feedback/interaction that

we find in modern Web information systems. In Hera-S we combine the existing strengths of Hera (i.e. its navigation modeling based on the semantics of the underlying data) and the Sesame capabilities for the processing of RDF-based data. Combining these trumps, it is possible to easily integrate data sources (e.g. for background knowledge), include external code and service elements, and to interfere with the data processing independently from the navigation. This enables a clean separation of concerns that helps in personalization and adaptation and in the inclusion of external data sources.

We are currently working on exploiting this facility for using aspect-orientation in the specification of user and context adaptation, and for the incorporation of business process and workflow specifications. We are also extending the available Hera design environment Hera Studio with the plug-ins for the SeRQL queries. Next to this work in extending the toolset, we also plan further investigation of the application of this solution in the domains of user profiles (within the framework of the IST MobiLife project) and multimedia and streaming content (within the framework of the ITEA Passepartout project).

8. REFERENCES

[1] Barna, P., Houben, G.J., et al: Navigation Design Support Using Reusable Navigation Templates. in: *IWWOST'05 International Workshop on Web Oriented Software Technologies*, Proceedings of the CAISE'05 Workshop on Web Oriented Software Technologies, Porto, Portugal, 13 June 2005.

[2] Brambilla, M., Ceri, S., et al.: Model-driven Specification of Web Services Composition and Integration with Data-intensive Web Applications, IEEE Bulletin of Data Engineering, December 2002

[3] Broekstra, J., Kampman, A. and van Harmelen, F.: Sesame: An Architecture for Storing and Querying RDF and RDF Schema. In *Proceedings of the First International Semantic Web Conference (ISWC 2002), Sardinia, Italy, June 9-12 2002, p. 54-68*. Springer-Verlag Lecture Notes in Computer Science (LNCS) no. 2342. See also http://www.openrdf.org/.

[4] Broekstra, J.: SeRQL: A Second-Generation RDF Query Language. Chapter 4 in *Storage, Querying and Inferencing for Semantic Web Languages.* PhD Thesis, Vrije Universiteit Amsterdam (July 2005). ISBN 90-9019-236-0. See also http://www.openrdf.org/doc/SeRQLmanual.html.

[5] Ceri, S., Dolog, P., et al: Model-Driven Design of Web Applications with Client-Side Adaptation, Lecture Notes in Computer Science, Volume 3140, Jan 2004, p. 201 – 214

[6] Chipman, A., Goodell, J., et al: Getty thesaurus of geographic names: editorial guidelines. Available at: http://www.getty.edu/research/conducting_research/vocabularies/guidelines/tgn_1_contents_intro.pdf. (2005)

[7] Fiala, Z., Hinz, M., et al: Design and Implementation of Component-based Adaptive Web Presentations. in: *ACM-SAC2004, ACM Symposium on Applied Computing (SAC)*, Nicosia, Cyprus, 14-17 March 2004, p. 1698-1704, 2004, ACM Press.

[8] Hobbs, J.R., Pan, F.: An ontology of time for the semantic Web. ACM Transactions on Asian Language Information Processing (TALIP) 3(1) (2004) p.66-85

[9] International Movie Database, Available at: http://www.imdm.com, 2006.

[10] Miller, G.A.: Wordnet: a lexical database for english. Commun. ACM 38(11) (1995) p. 39-41

[11] openRDF.org: home of Sesame, available at: http://www.openrdf.org/

[12] RDF representation of Wordnet, available at: http://www.semanticweb.org/library/

[13] Rutten, B., Barna, P., et al: A Tool for Presentation Generation in WIS. in: *WWW2004, The Thirteenth International World Wide Web Conference, Alternate Track Papers and Posters*, New York, USA, 17-22 May 2004, p. 242-243, ACM.

[14] SIMILE | RDFizers, Available at: http://simile.mit.edu/RDFizers/index.htm, 2006.

[15] Teknowledge Geographical Ontology, available at: http://reliant.teknowledge.com/DAML/Geography.owl:

[16] Thiran, Ph., Hainaut, J.L., Houben, G.J.: Database Wrappers Development: Towards Automatic Generation. in: *CSMR'05, Ninth European Conference on Software Maintenance and Reengineering*, Manchester, UK, 21-23 March 2005, p. 207-216, 2005, IEEE CS Press.