Hydragen

an implementation of Hera-S

Singh, Balpreet

Award date:
2007

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

MASTER THESIS

Hydragen: An implementation of Hera-S

by Balpreet Singh

Supervisors: Prof. dr. ir. Geert-Jan Houben
ir. Kees van der Sluijs

Eindhoven, August 2007

Contents

Abstract

This thesis, along with the Hydragen application, is the result of my graduation project at the Hera research group of the Technische Universiteit Eindhoven during the 2006-2007 academic year. The Hera research group focuses on research in the area of Web Engineering and Web-based Information Systems (WIS). Hera project aims to develop and implement a methodology for the design and engineering of WIS. Hera-S is the latest version of the Hera methodology on which the Hydragen implementation is based. It defines a layered approach based on modeling for the task of creating a Web Information System.

The Hera-S methodology distinguishes three parts in the design of a WIS: domain modeling, application modeling and presentation generation. The domain modeling step requires defining relations between various domain elements and provides the basis for queries in application model to operate on. The application modeling part is concerned with describing the dynamic navigation structure of a Web application. It uses RDF queries to retrieve elements from domain data and embed the results in the overall navigation structure to create an instance. Adaptation and presentation aspects of the application are modeled in a context model. The presentation generation part transforms the generated instance into a format suitable for viewing e.g. HTML or WML.

This document covers my work on implementation of a prototype framework to support the Hera-S methodology. The starting point for me was the description of the methodology and an initial version of Application metamodel, which dictates the structure of a valid application model. In addition to developing Hydragen (which is the name given to the engine implementing Hera-S), an extensive example was developed which demonstrated the features and the strength of the methodology as well the implementation.

Hydragen is completely implemented in Java as a Web application and assumes Sesame RDF store as database for domain model, context model, application model and the data content itself.

Preface

This master thesis concludes the studies I have been doing at the Department of Mathematics and Computer Science at the Eindhoven University of Technology (TU/e).

I would like to thank my supervisors Geert-Jan Houben and Kees van der Sluijs for their support and valuable input during my project. Their continuous guidance and constructive feedback was crucial for the success of this project. I would also like to thank Jeen Broekstra who supported me by answering the technical questions about the Sesame framework.

Finally I would like to thank my fellow graduate students located in room HG 5.38 for a nice and constructive environment in which it was pleasant to work.

Chapter 1

Introduction

1.1 Web Information Systems

Information Systems (IS), in general, are typified by being data-intensive and by the presence of heavy interaction with object systems (like business processes etc). Usage of the Web as front-end or as a data source of an Information System brings us to the realm of Web Information Systems (WIS). The Web introduces additional characteristics like information sources that are not under the direct control of WIS, heterogeneity and support for many (heterogeneous) users. So it is not just a front end. WIS is generally exposed to highly volatile information (distributed and heterogeneous), variety of system interfaces with varied capabilities and is expected to provide a personalized/adaptive end-user access. Some examples of Web Information Systems include Digital libraries, Reservation Systems, Virtual marketplaces, Electronic TV guides etc.

1.2 Semantic Web

As the complexity of Information Systems grow, it is virtually impossible to keep treating each system as a self-containing island. The strength of modern Information System is in interaction, collaboration and distribution. Would it not be convenient if your PDA agenda program could collaborate with the agenda of your dentist to find a common time slot and if accepted cancel the restaurant reservation you made a month ago. Here we are talking about potentially three different IS's from different vendors talking to each other. Semantic Web[18] is a step in making this possible. The technologies that support Semantic Web include information modeling languages (XML, RDF, OWL, ...)[8], transformation languages (XSLT, ...)[8] , reasoning tools (racer[3], pellet[10], ...), visualization tools (protege[2], swoop,...) and standard development APIs (Jena[1], Sesame[6],

Redland[4], ...). Using Semantic Web, end user applications can provide features like syntactic and semantic inter-operability, recommendation services, data integration from heterogeneous sources, personalization, querying and reasoning support.

1.3 Web Design Methodologies

As has been observed in fields like embedded systems, mechanical design etc, standardization of design and implementation methodologies and domain specializations are the first signs that a domain/field is maturing. Although the Web technologies and related development tools/languages have been around for more than a decade now, but the credit of bringing Web Design into the realm of Engineering goes to the development of Web Information Systems and the advent of Semantic Web. Until a few years ago, the Web was more or less an organically grown collection of HTML pages and media contents: mostly disconnected and without any serious business application. This trend has altered recently.

A Web interface has become a "must have" for a number of information systems such as project management tools, time writing tools, Enterprise Resource Planning packages etc. More and more companies are adopting the E-business model: using data-intensive Web sites to sell their products and services varying from books and travel tickets to electronic gadgets and house hold appliances. The immense amount of data that needs to be managed and presented by these Web Information Systems creates a unique challenge for creation and maintenance of these sites. For example, information must be updated, stored and retrieved for presentation in a user-friendly manner. The size of the data set makes keeping the look and feel of the site consistent an arduous job. Furthermore, the role of end users needs to be augmented, which allows Web sites to be responsive to the needs of individual users. This makes the situation more difficult, as appropriate customization support needs to be provided which allows the specialization of a general purpose of Web site towards the profiles of user groups or individuals.

Given the challenges mentioned above, most of the current Web Engineering Methods propose a layered approach to development of complex systems. Several dimensions of methods [11][16] used for development of data intensive Web sites are:-

- *Domain data structure*, which describes the information that is to be managed and presented by the target Web site.

- *Navigation*, which concerns the facilities that allow end users to browse and navigate across the target Web site. Most recent WIS not only deal with Navigation aspects but also support advanced application logic.

- *User Interface*, which describes the composition structure of the contents of Web pages that allow dynamic access to underlying data sources.
- *Presentation*, which expresses the look and feel (i.e. presentation styles and layouts) of user interface elements.
- *Customization*, which describes the way to specialize a general purpose of Web site towards the profiles of user groups or individuals.

The methods described in this section, address one or more of these dimensions. Although all these approaches have more or less the same basis, they differ in details and focus on slightly different issues. The key idea behind these approaches is use of Model Driven Architecture to support these aspects of Web Application design.

1.3.1 OOHDM/SHDM

In OOHDM [13], a hypermedia application is built in a four-step process supporting an incremental or prototype process model. Each step focuses on a particular design concern, and an object-oriented model is built. Classification, aggregation and generalization/specialization are used throughout the process to enhance abstraction power and reuse opportunities. The Object-Oriented Hypermedia Design Method (OOHDM) uses abstraction and composition mechanisms in an object oriented framework to, on one hand, allow a concise description of complex information items, and on the other hand, allow the specification of complex navigation patterns and interface transformations.

The SHDM (Semantic Hypermedia Design Method) design approach, is an extension of the OOHDM (Object Oriented Hypermedia Design Method) approach. SHDM is targeted for the design of Web applications for the Semantic Web, replacing the conceptual models of OOHDM with Semantic Web ontologies. The applications designed using the SHDM method keep a relational database in the backend and use ontologies to structure the metadata used for navigating the data.

1.3.2 OntoWeaver

OntoWeaver [15] is an ontology-based approach, which provides high level support for Web site design and development. It relies on the following major components to achieve its task: i) a site view ontology, which provides fine-grained modeling support for user interfaces and navigation structures of the target Web site, ii) a presentation ontology, which provides high level support for the specification of layouts and presentation styles for user interface elements, iii) a customization framework, which exploits the declarative specification of the target Web site and provides comprehensive customization support at design time as

well as run time, and iv) an OntoWeaver tool suite, which facilitates site designers to achieve tasks of Web site design, including defining ontologies, specifying the target Web site, and defining customization requirements.

A typical design process in OntoWeaver proceeds by iterating the following steps: i) designing the domain ontology; ii) specifying navigation structures and composing user interfaces; iii) defining layouts and presentation styles, and iv) expressing customization requirements.

1.3.3 OO-H

Object-Oriented Hypermedia[17] methodology uses an Object-Oriented approach to capture all the relevant properties involved in the modeling and implementation of Web Application Interfaces. The design process involves the construction of two additional views, complementary to those captured in traditional, UML-compliant, conceptual modeling approaches. These are, namely, the Navigation View, which extends a class diagram with hypermedia navigation features, and the Presentation View, in which the different elements regarding interface appearance and behavior are modeled by a series of interconnected template structures, expressed in XML. As a result, a language-independent front-end specification is obtained. From there a Web interface, which may be integrated with pre-existent logic modules and/or Web services, can be generated in an automated way.

1.3.4 WSDM

WSDM[14] is an audience-driven design method, meaning that the requirements of the intended users form the starting point for the method, in contrast with most design methods, which are data-driven. The different phases of the WSDM design process include Mission Statement, Audience Modeling, Conceptual Design, Implementation Design and final Implementation. The models of each of these phases have an explicit formalization which use ontologies.

1.3.5 Web-ML

WebML (Web Modeling Language) [21] is a visual notation for designing complex data-intensive Web applications. It provides graphical, yet formal, specifications, embodied in a complete design process, which can be assisted by visual design tools, like WebRatio. Note that this is currently the only commercial tool available in this domain.

This method has five models: structure, derivation, composition, navigation and presentation. These models are developed in an iterative process.

1.4 Document Overview

This document consists of seven chapters, every chapter starts with a summary. Most of the details of the implementation that was carried out in the project are explained in chapter 4 but other related information can be found in rest of the document.

Chapter 2 provides motivation for the thesis providing background information on the existing status at the moment project was started and explaining limitations of previous implementations of Hera. The overview of Hera-S methodology and the detailed application meta-model which forms the basis of the application model is explained in Chapter 3. The implementation details of Hydragen (which implements Hera-S) and the design decisions get covered in Chapter 4. Chapter 5 outlines step-by-step process of developing a Web application based on Hera-S. Chapter 6 dwells upon the performance analysis of the current implementation. Finally, the conclusion and ideas for future work are described in Chapter 7.

Chapter 2

Hera-S: Motivation

As part of the Hera program, a number of tools have been developed, these include multiple versions of Hypermedia Presentation Generator (HPG-XSLT, HPG-Java), HPG Builders (to create Hera Models visually) etc. These tools implement the basic Hera methodology and its variants. Hera-S is the next version of Hera methodology, which makes more use of the SeRQL language to manipulate the RDF data in the models. This chapter aims at describing the motivation for Hera-S.

Hera is a method for Web information systems (WIS) design that found its origins in an approach for hypermedia presentation generation[16]. It was also this focus on hypermedia presentation generation that gave the first engine complying with this method its name HPG (Hypermedia Presentation Generator). The method distinguishes three main models that specify the generation of hypermedia presentations over available content data. With a model for the content, an application model for application logic and hypermedia navigation structure, and a model for the presentation structure, the method enables the creation of a hypermedia-based view over the content. Originally, in the first generation of the method and its toolset, the models specified a static transformation from the content to the presentation. The engine that was compliant with this definition was based on XSLT and is therefore known as HPG-XSLT.

One of the characteristic aspects that HPG-XSLT supported was adaptation. As an illustrative example, figure 2.1 shows how different presentations could be produced by the engine out of a single design in which the "translation" to formats such as HTML, SMIL, and WML was dealt with.

Characteristic for the Hera models was not only their focus on user- and context-adaptation support, but also the choice to base the models on the Resource Description Framework (RDF) and RDF Schema (RDFS). The use of Web standards such as RDF and RDFS as a modeling paradigm facilitates easy de-

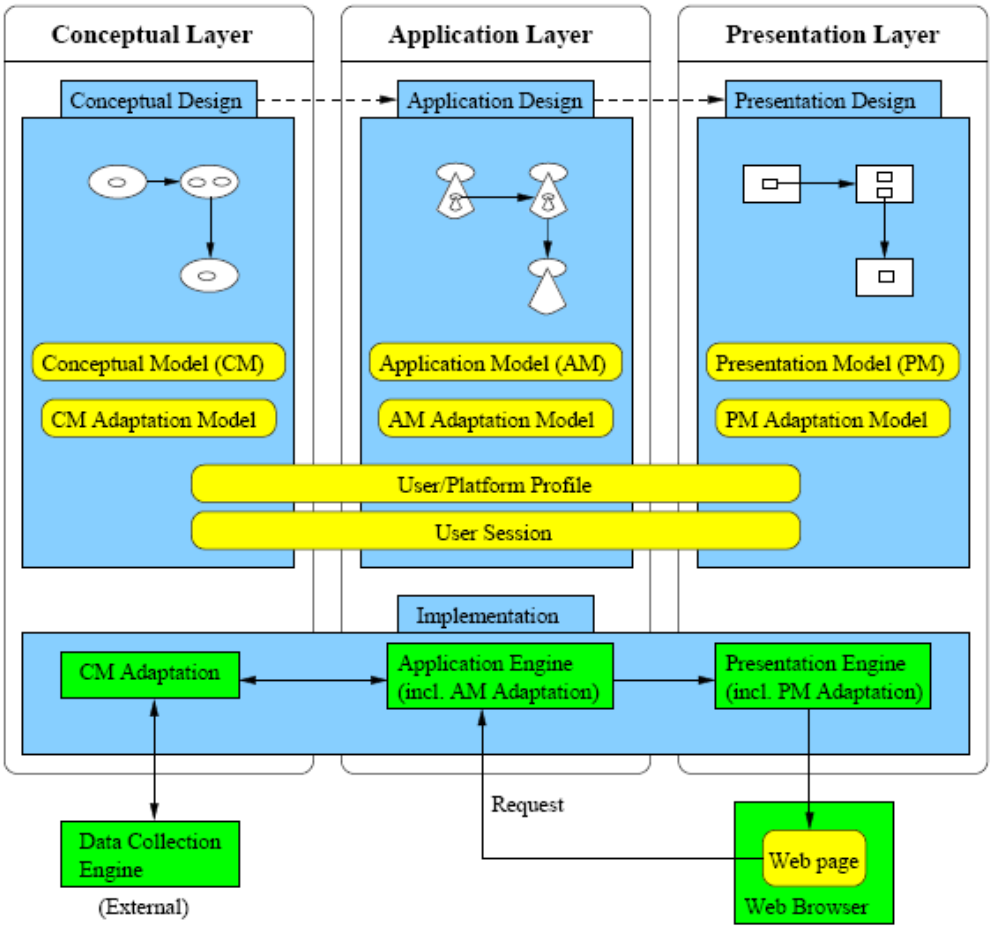

Figure 2.1: Hera Architecture

ployment on very heterogeneous data sources: the only assumption made is that a semi-structured description (in RDF) of the domain is available for processing. Not only is such a representation less costly to develop than any alternative, it also enables reuse of existing knowledge and flexible integration of several separate data sources in a single hypermedia presentation.

During the further research into the development of the method, the support was extended for more advanced dynamics. The first XSLT-based approach primarily transformed the original content data into a hypermedia document, with which the user could interact through following links with a Web browser. The subsequent engine version allowed the inclusion of form processing, which led to the support of other kinds of user-interaction. This Java-based version of the engine (HPG-Java) used RDF-queries to carry out actions specified as part of Form processing (select input elements etc.).

2.1 Hera Implementations: Limitations

HPG-XSLT and HPG-Java have both their advantages and disadvantages. In HPG-XSLT, focus was on static generation of hypermedia presentation, which can be customized by adaptation to user/platform profiles and by supporting various output formats. It is characterized by the use of XSLT stylesheets for the data transformations and by the full and static generation of a Web presentation. Due to this lot of flexibility is lost. In contrast, HPG-Java generates one-page-at-a-time in order to better support the dynamic Web applications.

The generation of the full presentation in HPG-XSLT requires usually a long time for computing the whole presentation. If one decides to deploy the resulted pages on a Web server this high computational time does not influence the system response time to a user. Given current state of technology, the user can browse the presentation at a incredible speed because there is no computation performed on the server. The generation of one page at-a-time in HPG-Java has as consequence a longer response time than for a presentation generated with HPG-XSLT. It has less of a startup-penalty, but this comes with a per page penalty because of the computation on runtime.

The resulted Web pages from HPG-XSLT can be deployed on any Web server. Due to its dynamic nature, HPG-Java can be deployed only on Web servers that support Java Servlets. HPG-XSLT has no support for user interaction besides simple navigation. The user of a generated presentation cannot influence the content of the presentation. HPG-Java does allow for more advanced forms of user interaction (e.g., forms) as a way to let the user influence the content of the presentation. This is an useful feature in case the application will be used for example as a shopping site or as a review system.

Since HPG-Java implementation used Jena as RDF API and Sesame for static content storage, it was limited in data integration possibilities. Besides, there were performance issues identified with it. From an application model point of view there were a few limitations as well. For example, queries included in the Application Model (AM) are associated to slices or slice relationships to select the data that will populate the next slice to be presented or to perform updates as a query side effect. There are four cases in which queries are used. In the first case queries are associated to slices to express user-independent updates (e.g., creation of a check-out trolley). In the second case queries are associated to forms (forms and form input fields are also slices) to express user-dependent updates (e.g., create order and add it to the trolley). In the third case, queries are used to get values for a form input field (e.g., select paintings names). In the last case, queries are used in form conditions, to enable/disable a certain form (e.g., if the user has already selected posters for all paintings, there is no painting poster left to be offered to the user for the next selection, and therefore the form is disabled). Basically, queries are not used to retrieve elements from the data

set itself. The link between content and application is via explicitly referring to elements from the content model. This is very limited and does not allow any constraints or conditions to be set for selection of the value of an attribute.

Another problem in the HPG was also that a specific vocabulary for domain, context and user profile was used. This had some drawbacks since this restricts the type of data that can be modeled and makes it format specific. This was improved upon in Hera-S by dropping vocabulary definitions for domain and context models. User is allowed to define his/her own domain and context model in RDFS/OWL which can be serialized in any of the formats such as XML, Turtle, Trix, N3 etc as long as they are supported by Sesame.

2.2 Sesame : RDF store

Sesame is a framework for storage, query and inferencing of RDF and RDF Schema. It can be used as a java library as part of any application for handling RDF or as a database server for remote access of repositories of RDF data. It supports highly expressive query and transformation languages: SeRQL and SPARQL. Although SPARQL is going to be a W3C standard, it is not really mature yet. SeRQL, in contrast, is more flexible and has been defined for use with Sesame. The server version allows native storage, main memory storage or a RDBMS as backend for RDF data. In addition, reasoning support allows inferred relations between elements to be used in queries.

Sesame 2.0 supports the concept of "context", which basically allows clustering of sets of RDF statements so that they can be handled together. It is possible to indicate for each statement or a set of statement which "context" they belong to. Each context is expressed as a URI. The main advantage of "context" is provenance; it allows identifying the source of a set of RDF statements and modifying only those statements if needed. In addition, it allows querying on statements that belong to a certain context, maintaining multiple versions of a set of statements etc.

As compared to Sesame 2.0, the first version of Sesame was inflexible in terms of query and storage model, which led to poor performance by construction. In Sesame 2.0 (which has be built from scratch) these limitations are not there; although in current beta version of Sesame 2.0 the observed performance was worse that of Sesame 1.0. This issue is currently being worked on by Aduna Software, responsible for maintenance of Sesame.

2.3 Project Assignment

The experience out of these HPG-based versions and the aim for further exploitation of the RDF-based nature of the models have led to a further refinement of the

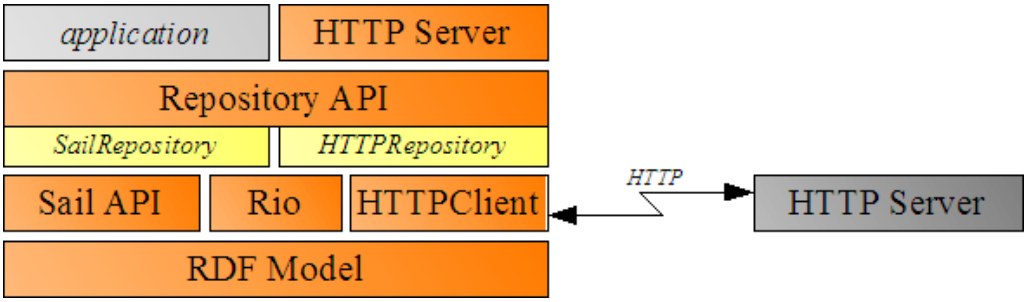

Figure 2.2: Sesame Overview

approach in what is termed Hera-S. The Hera-S compliant models do combine the original hypermedia-based spirit of the Hera models with more extensive use of RDF-querying and storage. Realizing this RDF data processing using the Sesame framework and its query language SeRQL caters for extra flexibility and interoperability. As compared to HPG-Java where data elements had to be extracted explicitly by name from a content model instance, Hera-S allows using query for this purpose making it less dependent on the exact name of a data element. Partially and optionally specified RDF paths and pattern matching for nodes in SeRQL queries makes development of AM independent of minor modification in Domain model (DM). Besides, RDF is a standard format, any tool that can generate RDF can add/modify content of AM/DM and CM (Context Model) on the fly. Similarly, any tool that can read RDF can consume the Application model page (an instance of AM which is emitted in RDF format, see 3.1 for details) and modify it. This contributes to the dynamic nature of the methodology and makes it easy to create extensions over the basic implementation.

Goals

The purpose of *Hera-S* project is to develop an engine (named *Hydragen*), which can be used for model based development of Web Information Systems as defined by the Hera-S methodology. This project is a follow-up of HPG-XSLT and HPG-Java projects. The Engine should behave as described in [24]. Globally, the Engine should meet the following constraints: -

- The product is supposed to be scalable and be efficient (this was not the case with the HPG-Java version of product).
- The Engine would be extendible for future changes or additions.
- Product must be developed in a modular manner to allow usage in different scenarios for example as a standalone tool as well as a Web server. Besides, the API for the product must be well defined. This is needed to provide flexibility and to support updating of models in real time without affecting

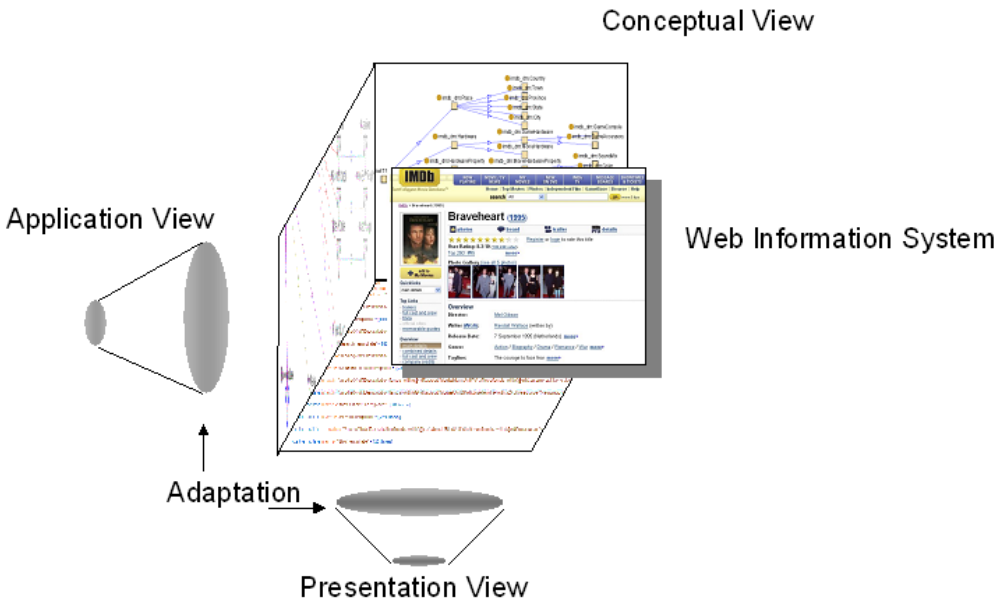

Figure 2.3: Hera Modeling

execution. In addition, standard API allows extending Hera-S with new features such as support for business logic.

- It should be possible for multiple users to access a Web application developed with *Hera-S* models without unexpected side effects.
- The product would potentially be considered to be developed into an open source project. This implies clean coding style and code documentation.

2.4 Project Benefits

Implementation of Hydragen is meant to be a proof of concept for Hera-S methodology. It is required to scale up for practical application both in terms of magnitude of data and application complexity and in terms of extensibility as well. In particular, Hydragen is expected to be used for further exploration such as support for Aspect-oriented adaptation, and support for presentation generation using engines like Amacont. In this context, a project (SeAL) has already been carried out in Vrije Universiteit Brussel with focus on implementing Aspect-oriented adaptation as described in Hera-S specification[24]. Note though this puts specific requirements on the implementation of Hydragen (which already covered above), the integration of SeAL and Hydragen is not part of this thesis.

Chapter 3

Hera-S: Design

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.

3.1 Methodology

Hera-S design methodology uses a Domain model as a starting point. The Domain model (DM) captures relations between various elements in the data content over which the Web Application is expected to operate. 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 also supports dynamic personalization and adaptation by maintaining context information, which is updated on the basis of user profile and interactions. The Context model (CM) describes the structure of the context information. For each request, an AM is instantiated to create an Application Model (instance) Page (AMP). Hence, AMP creation in Hera-S is pull-based, in contrast to HPG-XSLT, which does the whole instantiation of the AM at once. By navigating (link-following) and forms submission the user triggers the Hera-Ss feedback mechanism, which results in internal updating/querying of the

website navigation or context data and creation of a new AMP. AMP is not directly suitable for a browser but it can be converted to presentable form using direct AMP to HTML conversion (e.g using XSLT) or external engines such as AMACONT[20].

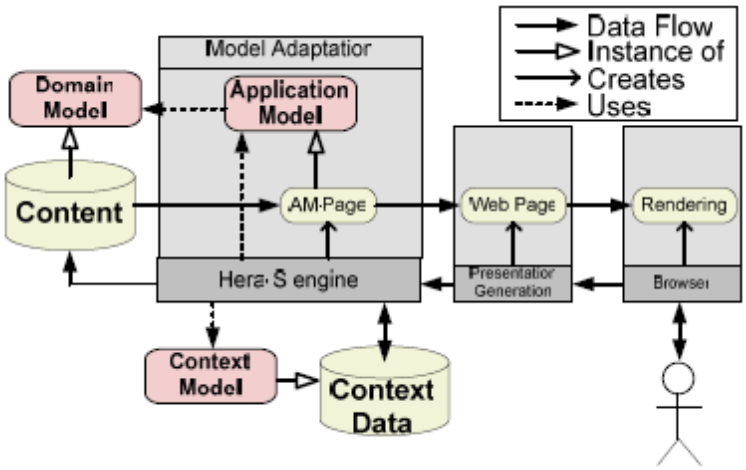

Figure 3.1: Hera-S Architecture

Hera-S methodology is explained in detail in *Hera-S* paper [24] and a chapter on *Hera* in a book on Web Engineering [12].

3.2 Application Metamodel

Web Applications built using *Hera-S* design methodology must comply to the Application metamodel (See AM-Metamodel). The *Hera-S* Application metamodel is subdivided in two main parts, namely a part for the basic constructs: *AmBasicElement* (units, attributes and relationship) and *ExtendedElement* (the rest of the am-constructs). An additional class is made for fundamental important notion of *Query*, that makes the connection between AM-elements and data in the database. The following sections go into depth on the several constructs in the AM-metamodel.

3.2.1 AmBasicElement

As shown in figure 3.2, main elements here are *NavigationalUnit*, *Attribute* and *Relationship*. Note that *FormUnit* is specialization of a *NavigationalUnit*, with specific subelements (i.e. *FormElements*). Similarly, *LogicUnit* is also a subclass of *NavigationalUnit* which is slightly different interpretation.

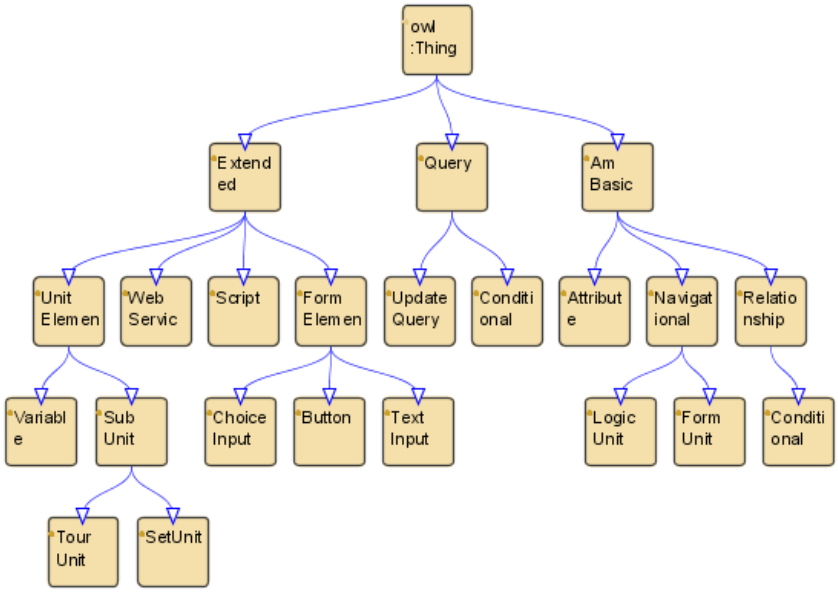

Figure 3.2: AM Elements

NavigationalUnit

A *NavigationalUnit* is a composite unit of information that can be linked together in a web of cross references to form a complete Web Application. One of the sub-classes of *NavigationalUnit* is called *LogicUnit*. In contrast to a *NavigationalUnit* which results in visual elements of a Web Application, *LogicUnit* contains only logical elements. Though structure wise *LogicUnit* is exactly the same a *NavigationalUnit*, it only makes sense to contain non-visual elements like *ConditionalRelationship*, Form queries, Update Queries etc in a *LogicUnit*. If visual elements (for eg *Attributes* etc) are encountered in *LogicUnit*, they will be ignored. *LogicUnit*s make provision for form processing without having to write scripts. It is especially useful when user input is needed to decide which unit should be displayed next. As soon as a *LogicUnit* is handled, the engine will automatically determine a uniquely defined referred *NavigationalUnit* and process that Unit. E.g. consider a *LoginUnit*: it contains a login form with input field for User and Password and a Submit button. As soon as the user fills in the input fields and submits the form, depending on whether his credentials are valid or not the engine must display the *LoginUnit* again or proceed to next unit such

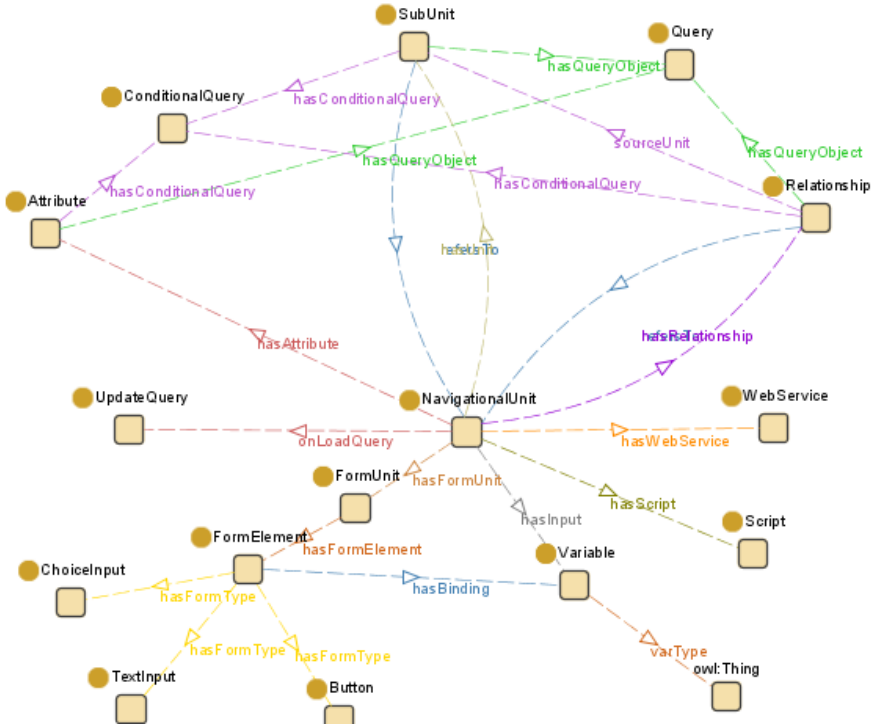

Figure 3.3: AM Elements Relations

as *SearchUnit*. This checking of credentials and choice of next unit to follow is typically something which gets handled in a *LogicUnit*.

A *NavigationalUnit* has the following possible properties:

hasAttribute: *NavigationalUnit*s can have multiple attributes. Attributes are the elements that are shown to the user.

hasInput: A *NavigationalUnit* can have multiple incoming variables. These variables are called the *InputVariable*s. The variables can be used for adaptation and personalization.

hasUnit: The *hasUnit* is used to specify some unit to be a subunit of another unit. The subUnit construction can be used to group related elements (e.g. having the same relationship).

The *hasUnit* property links to the *SubUnit* class. This is necessary to be able to specify which data should be selected. In some respect this is very similar to the *Relationship*, as also a query can defined, denoting which information should

be conveyed to the subunit. It differs, as subunits will be shown as part of unit, while relationships will cause navigation from some unit to another unit.

hasFormUnit: Indicates a *FormUnit* subunit. This kind of subunit does not necessarily need an incoming query, as its elements are usually only input-elements, and so there is (usually) no need to go back to the database.

hasRelationship: *NavigationalUnit*s are linked via *Relationship*s. The *hasRelationship* property is used to indicate the *Relationship* connected to the containing Unit. See more about relationships in Section 3.2.1.

onLoadQuery: The *onLoadQuery* indicates an action that should be executed during loading the Unit. The unit will be shown to the user after execution of the query. *onLoadQuery* is actually only one example of a query to be executed on a certain event. Many events can eventually be specified that will lead to execution of some action. These events may be added later as part of future work.

hasScript: *NavigationalUnit* can contain multiple scripts which are executed when a unit is loaded. See more about Scripts in Section 3.2.2

hasWebService: It is possible to involve a Web Service external to *Hera-S* application and display its results as part of the requested *NavigationalUnit*. *WebService* element is explained in more detail in Section 3.2.2

FormUnit

The *FormUnit* is similar to a normal unit, except that it also allows input-elements. These input-elements will be described in *FormElement* (Section 3.2.2).

Note that for the case of a *FormUnit* the relationship attached to it not only indicates the next unit, but usually also contains an update query to process the input of the form.

Attribute

An attribute is an element that is shown to the user; for example, the title of a specific movie. An attribute has the following properties:

hasQuery: The *Query* of an attribute indicates the element in the content database that constitutes the attribute.

The *hasQuery* property is the simple variant of the *hasQueryObject* and the value of this property consists of the query in the default query language, i.e. SeRQL.

hasQueryObject: The *hasQueryObject* is the complex variant of *hasQuery*. It points to a *Query* object (See Section 3.2.3). The *Query* object can be used to specify a more complex query type, e.g. by specifying a different query language, a conditional query or an update query.

hasConditionalQuery: A conditional query specifies that an attribute is only shown under specific conditions. The condition itself is specified by means of query and is specified in an if-then-else construct via the *ConditionalQuery* construct (see Section 3.2.3).

Relationship

*NavigationalUnit*s are linked via *Relationships*. Basically a *NavigationalUnit* has only one relationship, and clicking on any one of the visible elements within the unit will result in that one relationship being followed.

*Subunit*s of a unit can have a different relationship than their parent. In this way *SubUnit*s can be used to group elements that link to the same destination-unit.

Note that if you want to have a *Relationship* originating from one attribute only, you can create a containing unit for that attribute.

refersTo: The *refersTo* element indicates the target-unit of a *Relationship*.

hasQuery: A *Relationship* can have a *hasQuery* property. The result of the query is passed as a variable to the target Unit. This can for instance be used to indicate a specific instance for within the target Unit, so if the target Unit is about a "movie", the query can indicate about which specific movie the unit will be instantiated, for instance "The Matrix".

The *hasQuery* property in a relationship is for the rest equal to the *hasQuery* property in Section 3.2.1.

hasQueryObject: The *hasQueryObject* property has the same function as the *hasQuery*, and is equal to the *hasQueryObject* property in Section 3.2.1.

hasConditionalQuery: The *hasConditionalQuery* property has the same function as the *hasQuery*, and is equal to the *hasConditionalQuery* property in Section 3.2.1.

hasAssignment: In some situations, it is useful to pass a fixed value as binding to a variable to the target unit instead of using result from a query. The *hasAssignment* property allows support for such a case. For example, a Logout

relationship can refer to a LoginUnit unit or HomePage unit with User Variable bound to *GuestUser*.

sourceUnit: The *sourceUnit* indicates the starting-unit of a relationship. By default, the starting-unit will be the top-level unit of the current displayed unit. This means that the entire current page will navigate to the target unit. However, sometimes one might want to only let a part of the page navigate. This is similar to the HTML notion of frame.

The *SubUnit* that should navigate can therefore be indicated with the *sourceUnit* property. This *SubUnit* may be different from the *(Sub)Unit* that contains the actual relationship. This could for instance be utilized to create a menu-structure.

ConditionalRelationship: *ConditionalRelationship* is a subclass of *Relationship* which allows creating a conditional link to a *NavigationalUnit*. Multiple *ConditionalRelationship*s can be used to choose one of many relations depending on the attached condition. For example, depending on if a user login passes or fails, the next unit must be a search unit or user must be shown the login unit again. In addition to the properties mentioned for *Relationship*, *ConditionalRelationship* has the following elements:-

hasCondition: The *hasCondition* properties contains a boolean condition which must evaluate to true, in order to select the *NavigationalUnit* indicated in corresponding *refersTo* clause as the referred unit for the enclosing unit.

hasDefaultCondition: If none of the *ConditionalRelationship*s with *hasCondition* property is valid (their *hasCondition* does not evaluate to true), then the *ConditionalRelationship* with *hasDefaultCondition* clause is used to derive the referred unit. The value of *hasDefaultCondition* property itself is irrelevant. Note that though the *hasDefaultCondition* is optional, its always safe to provide one.

3.2.2 ExtendedElement

Extended elements are all non-vital elements of the *AM*, plus additional constructs that can be needed by the basic elements.

FormElement

*FormElement*s are input elements other than plain links. Examples of *FormElement*s are *TextInput*s, *Button*s, *ChoiceInput*s. Many others exist that are not subscribed here, but might be added to the am-metamodel later. As *Form*s are

very similar to the HTML and XForms notion of Forms, one could get inspired by their specification for additional form constructs.

hasBinding: The *hasBinding* property can be used to specify that the input for a certain element should be bound to a certain variable. An *AM* could for instance couple the user input for a "User Name" field to a variable "$name$". This variable can be used on submission of the form (so already in the relationship or update query), and in the units to come as a bound variable.

hasFormType: A *FormElement* is a generic element that has to be typed as one of the more specific types like button, choiceinput, textinput, etc (at least, as long as the formElement should not be abstract). This specialization can be done with the *hasFormType* property. A specialized *FormElement* can have additional specific properties.

Button: A button in general displays some text, and if clicked submits the form, according to the relationship attached to the form. Currently we keep button rather simplistic, but later this can easily be extended by attaching scripts to be executed when associated to a button.

ChoiceInput: A *ChoiceInput* can be used to let a user make a selection between items. *ChoiceInput* can in the presentation be visualized as radiobuttons, checkboxes, dropdownlist, or whatever variant the designer wants to use. However, which of these possibilities the designer wants the use is basically irrelevant in this phase and should not be specified here but in the presentation phase. An item can be added via the *option* property.

TextInput: *TextInput* elements stand for all possible Textual Input elements. Concrete examples are TextField, TextArea, PasswordFields, but also pure *Text Fields*, *Rich Text Fields*, etc. The concrete kind of those fields is again irrelevant at this point.

UnitElement

UnitElement is the general group of helper-elements that are needed can be used within a unit. This is an abstract class and is only used for hierarchical reasons but will not appear in an actual *AM*.

SubUnit: *SubUnit* is a construction to denote the Unit that is the subUnit of some other unit. The *SubUnit* construction can be used to denote a special kind of subUnit (they have a specific semantics that differs from regular subUnits),

namely a *SetUnit* or *TourUnit*. The *SubUnit* construction can also be used for identification of *SubUnit* by the *sourceUnit* property of Relationships.

The *SubUnit* class is necessary in order to specify the relationship between the containing unit and its referring subunit. This is expressed by a query. In many ways the *SubUnit* construct is very similar to the navigation-construct with the semantic difference that the referring unit will be defined as a substructure of the containing unit rather that specifying a navigation between the units. Consider for instance the Unit Movie that has a subUnit that will display the *leadActor* (including some additional information about that actor). The subUnit *leadActor* than typically refer to an Actor Unit, where the query in the subUnit specification will identify which specific actor to be shown (namely the lead actor of the movie displayed in the current Unit). Note that this query will typically has no more than one result.

Note that *SubUnit* is only a placeholder, so a *SubUnit* nor its descendants *SetUnit* and *TourUnit* - contain concrete elements. Instead they have a reference to the *NavigationalUnit* that will be used to provide content for the *SubUnit*.

SetUnit: Sometimes in a unit we want to contain subunits for each of the elements of a set. For example, in a *MovieUnit* we might want to provide information for all actors from the movie (and not just the lead actor). We specify similarly to a normal *SubUnit* with the difference that the query for the information to be displayed by the *SetUnit* will typically have a resultset larger than one. For every result in the resultset the *SetUnit* will display the defined attributes.

As an example consider the *MovieUnit* that should display information about every actor in that movie (e.g. name, photo, age). This can be done by creating a Unit called *Actor* that contains attributes for some actor (i.e. the name, photo and age attributes). Then within the *MovieUnit* a *SetUnit* is defined that refers to the Actor unit and as a query defines the actors that play in the movie currently displayed in the *MovieUnit*. This results that for every actor that plays in the current movie an Actor *SubUnit* is generated with the requested attributes.

TourUnit: A *TourUnit* is similar to the *SetUnit*. Only now the *SubUnit* will only display one element, namely the first element of the resultset of the *TourUnit* query. Furthermore it provides navigation primitives to navigate through the rest of the resultset in the order of that resultset.

This primitive can be used for example for a slideshow or a set process where the order of navigation is of importance (consider for instance a check-out procedure).

Variable: A variable is similar to a variable in a programming language. Currently there is only a simple notion of variable that is global throughout the *AM*.

Later it might be expanded to also discern local variables (i.e. local to the current unit and its subunits).

A variable has a name, a type and a default value. To give a variable a value it should be bound (regard the binding property that can be used in several places, see for instance Section 3.2.2 for binding a variable via a form). Binding a variable that already has a value results in overwriting it. The default value is used as the original binding of a variable, i.e. if it does not get reassigned then this value would be used.

Script

Current Web applications offer users a wider range of client-side functionality by different kinds of scripting objects, like Javascript and VBscript, stylesheets, HTML+TIME timing objects etc. Even though WIS methods like *Hera-S* concentrate more on the creation of a platform-independent hypermedia presentation over a data domain, and these scripts are often (but not always) browserplatform specific, we still provide the designer a hook to insert these kind of scripting objects.

The designer can specify within a scripting object whatever code he wants, as this will be left untouched in generating the *AMP*s out of the *AM*. Furthermore, the designer can add an *hasTargetFormat* property to specify one or more target-formats for format-specific code, e.g. HTML or SMIL. This allows later in the process to filter out certain format-specific elements if these are not wanted for the current presentation. The scripting objects can use the variables that are defined within the scope of the units. Scripting objects can be defined as an element within any other element (i.e. units and attributes). Furthermore, it can be specified if the script should be an attribute of its super-element or not (e.g. similar to elements in HTML that have attributes and a body).

Scripting objects are still very basic containers in Hera-S, and might later on be extended.

WebService

An application designer might want to use additional functionality that cannot be realized by a client-side object, but which involves the invocation of external server-side functionality. Therefore, we provide so-called *WebService* objects to support Web services in the AM. The use of a service object and the reason to provide support for it is similar to that of scripting objects. The designer is responsible for correctness and usefulness of the *WebService* object.

As an example, think of utilizing a Web service from a Web store selling DVDs in order to be able to show on a movie page an advertisement for buying the movies DVD. A service object needs three pieces of information:

- a URL of the Web service one wants to use,
- a SOAP message that contains the request to the Web service, and
- a definition of the result elements.

A service object declaration can be embedded as a part of every other element. If a unit is navigated to *created*, first the service objects will be executed. The results of the service object will either be directly integrated into the AM and treated as such, or the result can be bound to variables. *WebService* objects can use unit variables in their calls (meaning that variables will be initialized before *WebService* calls will be executed). This allows support for usecases such as determining the location of cinema hall in which user's favorite movie is playing and using the Google Map service to locate it on a map. As the zoom factor of the Map is increased more cinema halls showing more of the personal favorites can be displayed.

3.2.3 Query

Queries are used to connect AM-navigation structures to data in the content database. Queries are by default expected to be stated in the SeRQL language, as that is a mature query language. However, the *Hera-S* implementation will not depend on specific SeRQL or Sesame features, so also other RDF query languages or databases can be quite simple be implemented and added later on.

In order to already facility this the *queryType* can be used to indicate the query language if this differs from the default language. Furthermore special kind of queries can be specified, namely conditional and update queries.

queryBody

The *QueryBody* contains the actual query statement that will send to the configured database. Queries may contain variable statements denoted by the "$varname$" pattern (that is not in use by the Sesame-engine). This is not in the SeRQL specification but a Hera-S extension. The variables will be evaluated before the query is sent to the database.

queryType

The *queryType* denotes the query language of the query in the *queryBody*. If the concerning *queryType* is registered within the *Hera-S* engine it will deal with sending it to the correct engine, knowing that the engine can deal with the specified kind of queries, for example SeRQL or SPARQL.

ConditionalQuery

Conditional queries can be used to let the result of a query depend on another query or a boolean condition. This can be useful for selecting some element based on the context. Then you first specify a query for the context and based on the result you can specify one element or another one.

The conditional query has an if-then-else form. The "if" part contains the condition. If the "if" part produces some result, then "then" part of the if-query is executed. If there is no result from the "if" part the "else" part will be executed.

The *ConditionalQuery* is used for personalization and adaptation. For more extensive examples of how this powerful construct can be used see the bookchapter [12].

UpdateQuery

For the sake of adaptation we need to maintain an up-to-date context model. In order to do so, we need to perform updates to this data. For this, we have the functionality to specify an update query. Update query can for instance be executed upon certain events (see for instance Section 3.2.1).

An update query is typically an insert statement that inserts or modifies data in the context database that conforms the general context model. Note however, that *UpdateQuery* could potentially also be used to update the actual content (i.e. like in a content management system).

Chapter 4

Hydragen: Implementation

4.1 Software Architecture

Hydragen is composed of two main components: the *HydragenCore* and *HydragenWeb*. *HydragenCore* encapsulates the core engine functionality. For each request of a *NavigationalUnit* it is able to generate a new *AMP*. *HydragenWeb* on the other hand is a Java Servlet, which forwards requests to *HydragenCore* and transforms the generated *AMP* by applying *XSLT* transformation into *HTML* response. In that respect, *HydragenWeb* acts as the implementation for presentation model. In principle, the format of response is decided by *XSLT* specification used for presentation generation. Though within the scope of this project, *XSLT* has been used for the purpose of presentation generation, it is also possible to use more advanced system such as AMACONT [20], [23].

Setup

HydragenWeb is deployed as a Web application on a servlet container such as Apache Tomcat Server. In addition, the *Sesame* server and *Sesame* Web-client also need to be installed on a Web container (potentially different from the one used for *HydragenWeb*). The *Sesame* server must be loaded with *RDF* data on which the application is expected to operate along with the corresponding domain model and context model. The RDF models and content are loaded into a Sesame Repository which can later be access using a URI to the server and repository name over an HTTP Connection. All paths to application, data and context servers and initialization options can be configured by using the *Hydragen Configuration File* (for eg. imdb server setup file. See Appendix B.1. The path to hydragen setup file itself must be indicated in the Web application configuration file of *HydragenWeb* (i.e. web.xml). Sesame allows setting up of

several properties of the repository such as use of inference layer, type of storage (in-memory, native, or RDBMS), persistence etc. The details of installation instructions for *Hydragen* can be found in [22] while explanation about setup of Sesame RDF store as Server can be found at http://www.openrdf.com.

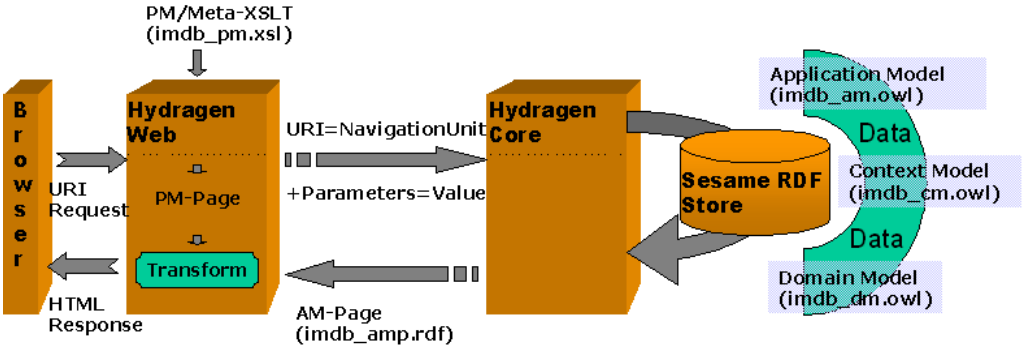

Figure 4.1: Hydragen Architecture

4.1.1 Static Structure

Deployment Diagram

As is shown in Figure 4.2, user access, *HydragenWeb* application and Sesame data and model server can all be on separate physical machines, only connected via network connection. In fact, in principle *DM*, *CM* and *Data Content* can also be hosted on separate Sesame Repositories. This scenario can be useful in case application is built over data source provided by a third party. But this scenario has not been explored in the context of this project since it requires support for distributed queries. Feasibility of distributed queries with Sesame has been studied earlier and presented in Distributed RDF Queries [19].

Hydragen access *Sesame* as *RDF* server for *DM*, *CM* and *RDF* Content. As for application model, an explicit choice was made to use *Sesame* in-memory repository instead of remote access variant due to the performance drop that was observed in the latter case (See 4.3 for detailed analysis).

Class Diagram

HydragenCore consists of the following packages (Appendix C.3) :-

- *nl.tue.heras.core*: This package contains the main controller class namely *CoreEngine* which is responsible for initialization of hydragen engine and handling of requests for a *Navigational Unit*. The *CoreEngine* class implements the interface defined in abstract class *IEngine*. It is also responsible for checking consistency of *Domain Content* against the *Domain Model*,

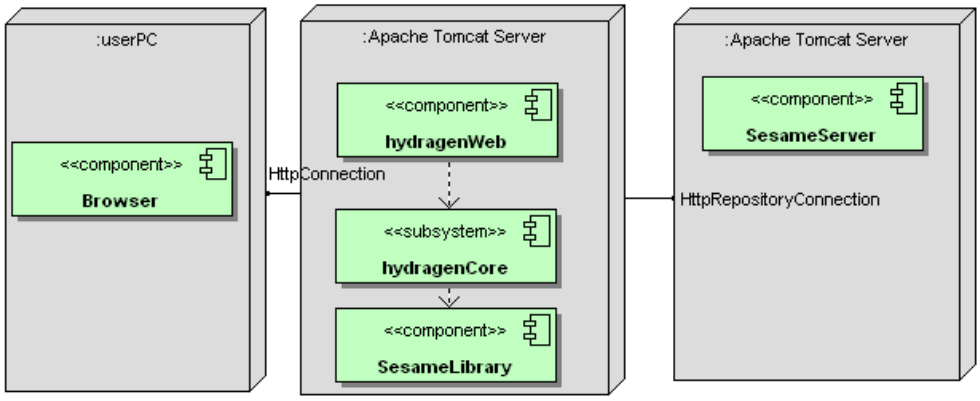

Figure 4.2: Hydragen Deployment

Context Data against the *Context Model* and *Application Model* against the *Application Meta-model*.

- *nl.tue.heras.parser*: The *parser* package contains the Application Model handle class i.e. the *AMParser*, which responsible for generating an instance of an Application Model Page based on the loaded Application Model. The *AMParser* uses SeRQL queries to identify and extract various AM elements (as explained in A), process Select, Construct and Delete SeRQL queries as part of various query elements and generate RDF statement based on the acquired results. *AMParser* uses the *DbManager* class which encapsulates operations on the Sesame RDF store, for all the query processing and access to *AM*, *DM*, *CM* and Content Repositories.

- *nl.tue.heras.backend*: The *backend* package contains classes such as *DbManager* which manages all accesses to the RDF store including setup, querying, adding and deletion of statements.

- *nl.tue.heras.datatypes*: This package contains the basic datatypes that are used within *Hydragen*. Classes such as *DbHandle* provide a "handle" to the RDF Repository and can be used to instantiate AM, DM, CM and Content repositories. The exact attributes of a repository are stored in an instance of the class *DbInfo*. *SysConfig* provides operations to parse *Hydragen Configuration File*.

- *nl.tue.heras.utils*: Utility classes such as exception related classes and *H2Utils* belong to the *utils* package. There are three main exception types supported within *HydragenWeb* in addition to the standard Java exception classes:-

- **AMException**: This exception is raised for any inconsistency detected in the Application Model. Though most of the errors in Application Model should be detected during consistency check with respect to Application Meta-model, some explicit checks are made within the *AMParser*. These include situations such as when the data queries within an AM are malformed or if the value of an input variable is not defined for a Unit etc.
- **CfgException**: Hydragen Configuration File syntax errors result in this exception to be raised. In addition, if paths to RDF models are incorrect then CfgException is reported.
- **DbException**: All errors encountered during access of RDF store are reported as *DbException*, including the ones generated by Sesame.

4.1.2 Dynamic Structure

There are two distinct phases in the execution of *Hydragen*. The initialization phase which is needed only when *HydragenWeb* is installed as a Web application and the execution phase which involves the request-generate-respond cycle.

As part of Servlet initialization step (Appendix C.1), *HydragenWeb* invokes the initialization function *initEngine()* of the *HydragenCore* library. Depending on the setting in the server configuration file, *HydragenCore* either loads various models from a local file or the engine (*HydragenCore* is configured to setup an *HttpRepositoryConnection* to a given remote repository address. *HydragenWeb* then loads the presentation model which in our case is an *XSLT* file with some predefined variables.

In the execution phase (Appendix C.2), the *HydragenWeb* servlet receives an *HttpServletRequest* everytime user executes a navigation action (e.g. click a link, submit a form etc). *HydragenWeb* implements both GET and POST methods as required by HTTP protocol for HTML response and for form submission. The GET method is meant to be used for calls with no side effects, for example modifications to a database etc. and also has a limit on amount of data that can be passed as parameter value. In contrast, POST method is usually used for forms which can send quite some data to the server. Internally, the *doGet* method invokes the *doPost* method. The request is in the form of a *URI* along with a set of parameters. The parameters passed with the request are retrieved, and stored in an internal data-structure (a *TreeMap* which keeps the parameter names sorted in alphabetical order to make sure that the AM variables with longest match are replaced) along with their corresponding values. The *URI* of the requested *NavigationalUnit* comes as value of a predetermined *HttpServletRequest* parameter - *URI*. Since during generation of HTML response, some encoding is done for URI's that are generated as part of response, the parameters need to

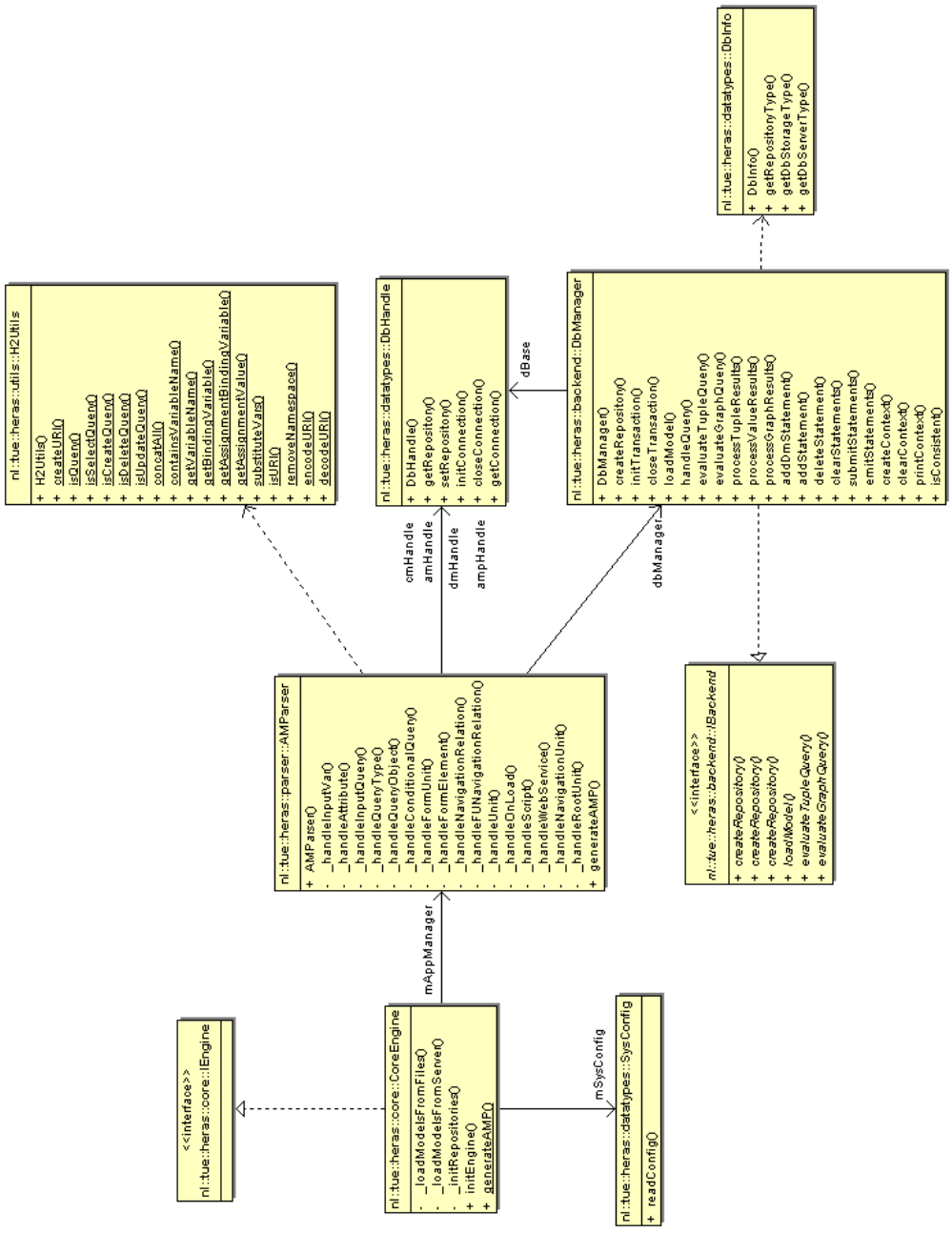

Figure 4.3: Hydragen Class Diagram

be decoded before being stored and passed on to the *CoreEngine* for request handling. Encoding of URI's is done to remove any special characters like angle brackets that are not allowed to occur in a URI. A typical example of a URL request for NavigationalUnit would appear as follows:-

```
http://localhost:8080/herasWeb/HydragenWeb?
URI=http://wwwis.win.tue.nl/~hera/Hera-S/imdb_am.owl^LoginUnit
```

The *generateAMP()* method from *CoreEngine* class is responsible for processing the request for a specific *NavigationalUnit* and generating the corresponding *Application Model Page* in a session specific file. The *CoreEngine* forwards the *generateAMP* request to the *AMParser* class which initializes query transactions with the *AM*, *DM* and *Content* repositories before processing the request further. *AMParser* then initializes the *AMP* output file as an *RDF* file. Although a request is always for a specific *NavigationalUnit* or *LogicUnit*, application metamodel allows a *NavigationalUnit* to contain more *NavigationalUnit* and in order to distinguish between handling of the root *NavigationalUnit* and the internal *NavigationalUnit*, the *handleRootUnit* method is called. As can be seen in the Activity Diagram, figure 4.4, root *NavigationalUnit* is processed only once for per request, while internal *NavigationalUnit* can potentially be processed several times depending on the Application Model.

Since *Hydragen* support creation of new nodes, deletion and update of statements, the statements that are generated/modified during processing of a *NavigationalUnit* are captured in a set of internal lists: add_list, delete_list, amp_list. As a last step before dumping the *AMP* to a file, these modifications are submitted. The addition and deletions (from add_list and delete_list) are submitted to the Data RDF store, while the *AMP* statements (from amp_list) are first stored in the session specific context of the *AMP* Repository. After that the contents of that session context of *AMP* repository are then dumped to file in RDF/XML format. The reason for dumping it to a file are two folds: firstly it allows easy XSLT processing which takes a XML file as starting point and another is it facilitates writing XSLT by looking at generated *AMP*.

Handling a *NavigationalUnit* involves processing all *AM* elements contained within it which includes:-

OnLoadQuery

The *onLoadQuery* is executed first during processing of a *NavigationalUnit*. Since it has to be an *UpdateQuery*, it does not generate any *AMP* statements. *OnLoadQuery* can be used to handle submit queries from *FormUnit* from previous *NavigationalUnit* or do some preprocessing before displaying the next Unit (for eg. updating the visit counter).

Input Variables

Variables play an important role in supporting dynamism and reuse in *Hydragen*. A *NavigationalUnit* can generate completely different result for a different values of certain variables. All the queries specified in an *AM*, can refer to a variable.

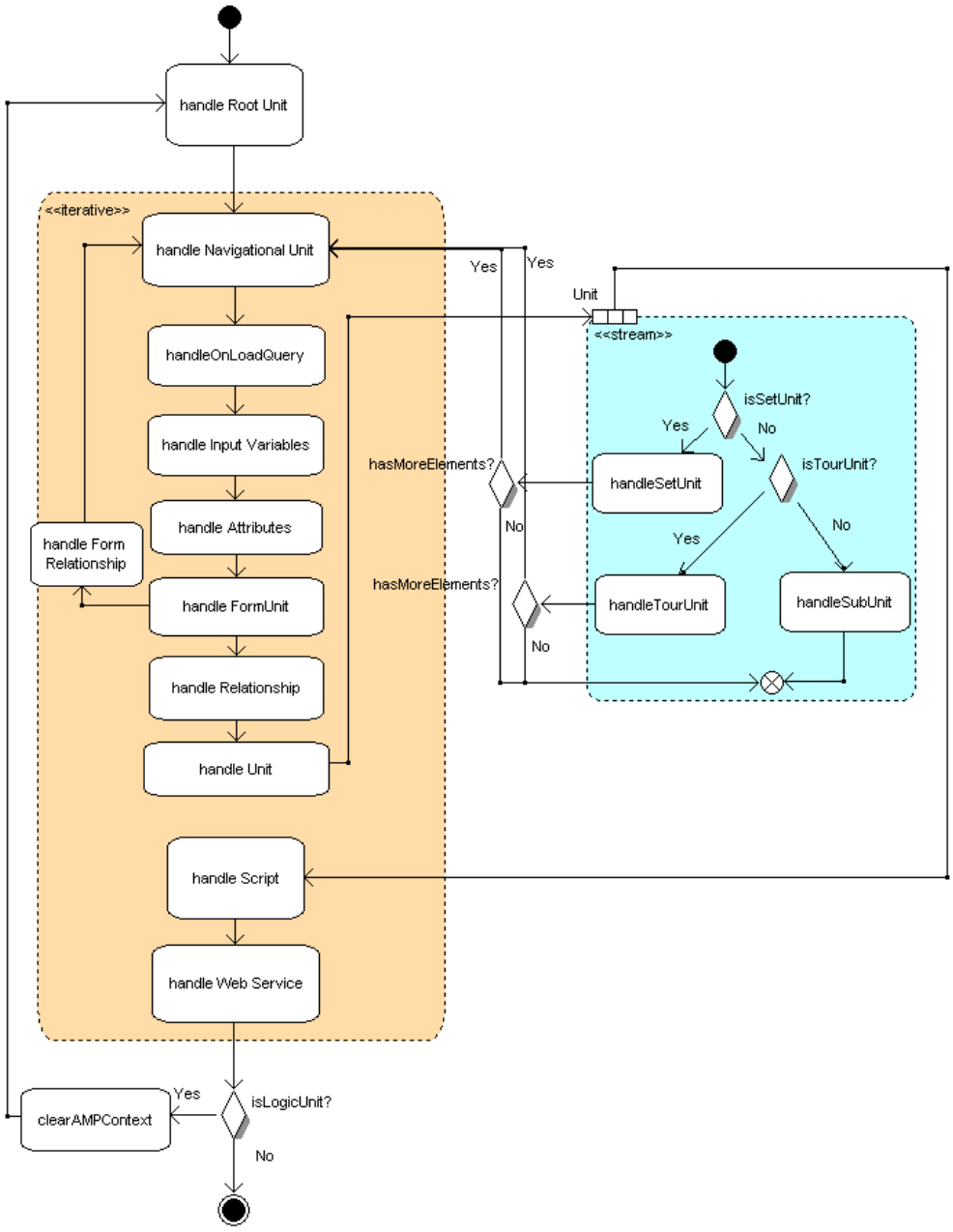

Figure 4.4: Handle Root Unit

This variable can either be initialized with a default value or from the result of another query. It can also be passed as a request parameter from one *NavigationalUnit* to another. Variables have a global scope in *Hydragen*. They are never deleted, their values can just be over-ridden with new ones. The consequence of this choice is described in Section 5.2.

After on-load queries have been processed, next step is to check if all input variables defined for the requested *NavigationalUnit* have a valid value and if not assign the *DefaultValue* as defined in the *AM*. In case, an input variable is undefined and no default value has been defined for the variable, an *AMException* is generated.

Attributes

As described in Section 3.2, attributes are individual elements of information that need to be displayed as part of the generated Unit. Attributes are allowed to be constants, result of a query or result of conditional query. Handling attributes requires determining the label, the value and the mime-type of an *Attribute*. An *Attribute* can occur as part of root *NavigationalUnit* or as part of *NavigationalUnit* referred by a Sub/Set/Tour Unit. In case of Set/Tour Unit, *AMP* statements must be generated for attribute of each element in a Set/Tour Unit.

Form Unit

Hydragen supports generation of a unit containing a *FormUnit*. *FormUnit* is a sub-class of a *NavigationalUnit* which implies that it can contain all elements allowed in *NavigationalUnit* in addition to some *FormElement*s such as *Button* elements, *Choice* elements, *Input* elements etc. All these elements are generated as part of the *AMP*. In addition, a relationship within a *FormUnit* needs to be handled differently as compared to that of a *NavigationalUnit*. In case of a *NavigationalUnit*, a relationship is translated to a hyper-link to another *NavigationalUnit*. For *FormUnit*, usually a query needs to be run as part of submit action of the form. Since only after the form is submitted are all the input, choice parameters known, running this query only makes sense when the target *NavigationalUnit* is processed. In order to facilitate this, the form relationship query is encoded as part of the *URI* that is generated for the relationship to next unit. When the form is submitted, the HTTP request gets the *Query* variable as a request parameter and gets handled as rest of the parameters. Processing of this query is the first action that is taken by *Hydragen* engine even before handling the root *NavigationalUnit*.

Relationship

As described in Section 3.2.1, Navigational *Relationship* uses the *refersTo* property to indicate which *NavigationalUnit*, current *NavigationalUnit* or *FormUnit* points to. In order to support choice between multiple relationships depending on boolean conditions, *AM* supports *ConditionalRelationship* which has the *hasCondition* and *hasDefaultCondition* properties. Depending on the boolean value of the conditions specified in *hasCondition* property, the corresponding *refersTo*

property is used to determine the linked *NavigationalUnit*. In case none of the conditions evaluate to true, the unit specified by the default condition is used.

A *Relationship* can also contain a *Query* object, the result of which is used to determine the value of a binding variable to be encoded as part of the link. This variable is then later used to instantiate the referred *NavigationalUnit*. It is also possible to pass a fixed assignment of a variable using the *hasAssignment* property. For example, in case of an application based on IMDB movie database, it is possible to generate a actor page which contains a set of movies in which a particular actor acted and each such movie name links to the corresponding movie page. This would be possible using the *hasAssignment* property in the *AM*.

Finally, it is also possible to model the target frame of a relationship by defining the *sourceUnit* property, in case of multi-frame setup (see section 3.2.1).

SubUnit

A *NavigationalUnit* is allowed to contain *SubUnit*s which refer to another *NavigationalUnit*. A *SubUnit* can be embedded into the parent *NavigationalUnit*. From implementation point of view, the *Query* element in a *SubUnit* is executed to retrieve the value of binding variable. The referred *NavigationalUnit* is then processed with retrieved value as value of one of its input variables.

As explained in Section 3.2.2, *SubUnit* has two variants namely: *SetUnit* and *TourUnit*. While in case of *SubUnit* the query is expected to return a single value, for *SetUnit* and *TourUnit* multiple results are possible. For eg. in case of the IMDB example, the set of movies in which an Actor played role can be modeled as *SetUnit*. Similar to *SubUnit*, *SetUnit* (since its a subclass of *SubUnit*) also refers to a *NavigationalUnit* which basically defines the attributes and relationship for every element of *SetUnit* query result. In order to keep the uniqueness of all generated *AMP* statements, an auto-incrementing postfix is used for all elements in the referred *NavigationalUnit*. *TourUnit* is handled similar to the *SetUnit*. The ordering of elements in the TourUnit is decided in the presentation generation step.

ScriptObject

Hydragen supports scripts by just instantiating any known variables from the "current" environment and pass the replaced *ScriptObject* as it is.

WebService

WebService's in general can be accessed using one for the following two ways:

- As Representational State Transfer (REST) requests

- As SOAP requests

Currently, *Hydragen* only supports the REST request based access. This is due to the ease of implementation since REST uses HTTP as its underlying transport. REST commands and options are passed to a WebService by using URL query parameters; the WebService returns an XML document containing the result. This process makes it possible to invoke WebService using a URI. For example in order to retrieve the price and related information about a DVD of Movie 'Titanic' from Amazon, following request URI would be sufficient:

```
http://webservices.amazon.com/onca/xml?
      Service=AWSECommerceService&
      SubscriptionId=xxxx&
      Operation=ItemLookup&
      ItemId=B00000JLWW
```

with the xxxx replaced by an Amazon Subscription ID (a unique subscription ID is needed before using any WebService; this can be acquired by registering at a Service Provider's site). This would return the following XML as result:-

```
<?xml version="1.0" encoding="UTF-8"?>
<ItemLookupResponse
       xmlns="http://webservices.amazon.com/AWSECommerceService/2005-10-05">
  <OperationRequest>
    <HTTPHeaders>
      <Header Name="UserAgent"
              Value="Mozilla/5.0 ..."/>
    </HTTPHeaders>
    <RequestId>1695HNYXPWDQZXAD7FQ8</RequestId>
    <Arguments>
      <Argument Name="ItemId" Value="B00000JLWW"/>
      <Argument Name="Service" Value="AWSECommerceService"/>
      <Argument Name="SubscriptionId" Value="XXXX"/>
      <Argument Name="Operation" Value="ItemLookup"/>
    </Arguments>
    <RequestProcessingTime>0.0367751121520996</RequestProcessingTime>
  </OperationRequest>
  <Items>
    <Request>
      <IsValid>True</IsValid>
      <ItemLookupRequest>
        <ItemId>B00000JLWW</ItemId>
      </ItemLookupRequest>
```

```
    </Request>
    <Item>
     <ASIN>B00000JLWW</ASIN>
     <DetailPageURL>
        http://www.amazon.com/gp/redirect.html
        %3FASIN=B00000JLWW%26tag=ws%261
        code=xm2%26cID=2025%26ccmID=165953%26
        location=/o/ASIN/B00000JLWW%253FSubscriptionId=XXXX
     </DetailPageURL>
     <ItemAttributes>
       <Actor>Lewis Abernathy</Actor>
       <Actor>Suzy Amis</Actor>
       <Actor>Jason Barry</Actor>
       <Actor>Kathy Bates</Actor>
       <Actor>Nicholas Cascone</Actor>
       <Creator Role="Primary Contributor">Winslet, Kate</Creator>
       <Manufacturer>Paramount</Manufacturer>
       <ProductGroup>DVD</ProductGroup>
       <Title>Titanic</Title>
     </ItemAttributes>
    </Item>
  </Items>
</ItemLookupResponse>
```

Hydragen extracts the *Items* element from the response XML and adds it to the *AMP* as value of the WebService node. It is the responsibility of the Presentation Model to retrieve the relevant information from the XML embedded in the *AMP* and display in the suitable form.

Query

Queries are the backbone of an *AM*. As explained in 3.2, *Hydragen* supports multiple types of queries: simple query, conditional query and query objects. Besides update query, regular queries retrieve information from the application data as well as context data store and pass it around. The way this is achieved is by binding the result of a query to a binding variable. In *Hydragen*, the binding variable is not explicitly defined but extracted from the query itself. The variable-value binding retrieved from the SELECT clause of a SeRQL query is used to define a variable-value binding for the *AM*. This only works since all Select queries in the *AM* are expected to have single variable. Before a query is executed, all variables are substituted. Note at this moment only SeRQL queries

are supported in *Hydragen*, though extending the support to other RDF query languages is rather straightforward.

LogicUnit

As described in Section 3.2, *LogicUnit* is a subclass of *NavigationalUnit*. Since the purpose of *LogicUnit* is only to handle form queries and similar actions which do not generate visual content, at the end of processing a *LogicUnit* the generated *AMP* is cleared and the engine navigates to a unique next *NavigationalUnit*. Since *Hydragen* must browse to the next *NavigationalUnit* automatically in case of a *LogicUnit*, application developer must take care that there is only one unique choice of *NavigationalUnit*. There is no guarantee, which one will be picked in case multiple relations turn out to be valid. This is due to the fact that several links are supported by using *ConditionalRelationship* and if multiple of them are valid at the same time the first one encountered is taken.

4.2 Supporting technologies

- *Sesame*: As described in Section 2.2, Sesame is a framework for storage, query and inferencing of RDF and RDF Schema. The choice for Sesame for *Hydragen* is motivated by following factors:-
  - Support for SeRQL which is a powerful and easy to use RDF query language.
  - Support for "Context" for provenance.
  - Support for RDF inferencer
  - Close working relationship between TU/e and Aduna Software, which maintains Sesame.

  Sesame offers a high level Repository API to that offers a large number of developer-oriented methods for handling RDF data. The main goal of this API is to make the life of application developers as easy as possible. It offers various methods for uploading data files, querying, and extracting and manipulating data.

  Underlying the Repository API is the Storage And Inference Layer (Sail) API which is a low level System API (SPI) for RDF stores and inferencers. Its purpose is to abstract from the storage and inference details, allowing various types of storage and inference to be used. The Sail API is mainly of interest for those who are developing Sail implementations, for all others it suffices to know how to create and configure one. There are several implementations of the Sail API, for example the MemoryStore which stores

RDF data in main memory, and the NativeStore which uses dedicated on-disk data structures for storage.

- *Apache Tomcat*: Although it is primarily a concern for end Application developer, it was required to choose a suitable web server for the purpose of demonstration of Hydragen as well as for the development and test environment. Speed, scalability, support for Java Servlet and ease of deployment were the key factors that influenced the decision here. The candidates that were considered for this include:
  - Apache Tomcat
  - Resin
  - Sun Java Application Server

  Out of these, during development Apache Tomcat was the preferred choice due to its ease of deployment, native Java support and integration in Eclipse development environment. Some of features supported by Apache Tomcat include native support for Java Servlet, XSLT transformation, etc. It is also the supported Web container for Sesame.

- *Java*: *Hydragen* has been implemented in Java mainly due to the extensive amount of support libraries available in Java for example Sesame API, Pellet, Saxon etc.

- *Pellet*: Pellet[10] is an open source, OWL DL reasoner in Java, originally developed at the University of Marylands Mindswap Lab. It ensures that an ontology does not contain any contradictory facts. The OWL Semantics standard provides the formal definition of ontology consistency used by Pellet. In the context of Hydragen, Pellet is used to determine if all the models are compliant to their schemas and are consistent. Pellet was selected mainly because it was the only freely available reasoner with a decent quality. It can, in fact, be used in future to debug and repair models.

- *Xerces*: Xerces2[9] Java Parser is an XML parser, which is used to parse the XSLT presentation model and add some XSLT variable definition nodes to parsed DOM3 structure. The updated version of XSLT is then written out as an XML file. Xerces is also used to parse Hydragen configuration file.

- *Saxon*: SaxonB[5] is a XSLT 2.0 processor whose native API is used to apply XSLT transformation on generated *AMP* to create HTML output.

- *Beanshell*: BeanShell is a small, embeddable Java source interpreter with object scripting language features, written in Java. BeanShell dynamically

executes standard Java syntax and extends it with common scripting conveniences such as loose types, commands, and method closures like those in Perl and JavaScript. Beanshell is used to evaluate the boolean condition expressions and update query expressions in *AMParser*. An external interpretor like BeanShell is needed since these boolean expressions are extracted from the *AM* as strings, and it is not possible to evaluate strings as Java expression in Java.

- *log4j*: With log4j it is possible to enable logging at runtime without modifying the application binary. The log4j package is designed so that these statements can remain in shipped code without incurring a heavy performance cost. Logging behavior can be controlled by editing a configuration file, without touching the application binary.

  One of the distinctive features of log4j is the notion of inheritance in loggers. Using a logger hierarchy it is possible to control which log statements are output at arbitrarily fine granularity but also great ease. This helps reduce the volume of logged output and minimize the cost of logging.

4.3 Design and Implementation Choices

During the course of the project, quite a few issues were found. In order to overcome these issues, some design decisions and implementation choices were made. The section below give a background on the modifications made in order to solve the related issues. In addition, it provides insight into the working of the *Hydragen*.

4.3.1 Metamodel enhancements

At the start of the project, the initial AM metamodel was given based on the HeraS paper[24]. During the creation of example application, I discovered some limitations that needed to be improved. After discussion with the supervisors a selection of modifications were accepted and brought back into the AM metamodel. The metamodel enhancements which came out of these discussions explained below:-

- **LogicUnit**: Initial version of Application meta-model did not have the concept of *LogicUnit*. During implementation of *FormUnit*, it was realized that since the processing of the form query needs the information entered by the user, the query must be executed in the following unit. Moreover, it is possible that the choice of following unit depends on the result of this query for example in case of a login form, if login succeeds user goes to the content page otherwise he must be requested to try to login again or

register. All this "logic" processing does not completely comply with the idea of *NavigationalUnit* which is intended to capture navigational aspects of the application. Hence, the concept of *LogicUnit* was introduced which although is a sub-class of *NavigationalUnit*, but it is not meaningful for it to contain any visual elements. It should typically be used to model form query, conditional relationships, update queries etc.

- **Conditional Relationship**: In order to handle situations where choice between referred *NavigationalUnit* has to be made based on a certain condition, the concept of conditional relationship was introduced. The *hasCondition* and *hasDefaultCondition* properties were added to *ConditionalRelationship* to specify boolean conditions, for choosing between multiple *NavigationalUnit*'s if one of the condition evaluates to "true" or the default one if none of them is valid. *ConditionalRelationship* itself is a sub-class of the *Relationship* concept.

- **Relationship**: Original Application meta-model allowed only queries to define variable bindings for referred *NavigationalUnit*. During the project, additional *hasAssignment* property was added for explicit variable assignment.

- **Boolean Expressions in Conditions**: Boolean conditions occur in multiple places in an *AM* such as *ConditionalRelationship* and *ConditionalQuery*. According to the original specification, a condition had to be formulated as a *Query* which the connotation that if the query generated a result then condition evaluates to true otherwise false. During implementation of IMDB database example with *Hydragen*, it was soon evident that in some situations the definition of condition needs to be extended to include boolean expressions. For example, in order to check if user wants to search for a movie or an actor in the *SearchUnit*, the value of choice made by the user can be stored in a *searchCategory* variable and later the value of this variable can be used in a *ConditionalRelationship* to decide which unit to refer to (since the *ActorResultUnit* would be different from a *MovieResultUnit*).

- **Variable**: The *hasDefaultValue* property was added to *AM* variables to indicate default values for a variable if it does not have a value at the moment of use.

4.3.2 Implementation related

How is validating models supported?

The models loaded in the engine need to be checked for validity. This requires syntactic check as well as semantic consistency with provided schemas. Hence,

an Application Model must correspond to the AM-metamodel, the Domain data must correspond to the Domain model and the Context data must be consistent with its Context model. For syntax check, *Hydragen* relies on Sesame to report any errors in the models or in Domain content. On the other hand, semantic consistency is checked by using *Pellet* as an OWL-DL reasoner. Since, *Pellet* needs to load the models and corresponding data before it can inform it is consistent or not, consistency check is only implemented for data loaded from a file in *Hydragen*. Hence, only *AM* is checked against the AM-metamodel since the rest are pre-loaded and are only accessible via the Sesame RDF server.

How are Data and Model repositories setup?

Since the Application Model can have queries, which go across Context Model and Domain Model, and since Sesame currently does not allow cross-repository queries, it is required to use same Sesame Repository at least for Context and Domain models. The Application Model itself can be maintained in a separate repository. Within the repository for *CM* and *DM*, the Sesame "context" information can be used to localize the scope a query or an update request.

In some cases, it could be required to have separate *DM* and *CM* , for example if we don't have control over data set itself. If required this restriction can be overcome by using distributed querying or introducing a mediator as described for example in [19].

The generated *AMP* is stored in a separate repository and a new context is generated for every new session in order to separate data from multiple users. The reason for using separate repository is performance. Normally, the *AM*, *DM* and *CM* repositories can be setup with inference in order to facilitate query construction. But this is has consequence on overall speed of the system. In case of *AMP*, repository is just a temporary store before its dumped to a file. There are no queries executed on the *AMP* itself hence it does not make sense to turn on inferencing for *AMP*. Hence, choice was made to setup *AMP* in a separate repository. Note that in future, it might be useful to directly access *AMP* from repository for presentation generation using extended XSLT. (See Section 4.3.6). It might be needed to reconsider this choice then.

How is inferencing used to handle a request?

RDF/OWL inferencing uses RDF and OWL entailment rules to extend a set of RDF/OWL statements with additional logically derivable statements. For example, if an OWL class A has property B and another class C is a sub-class of A, then property B also holds for C. Hence, the statement that the property B holds for C can be derived by RDF entailment and added to the original set.

As is evident from the structure of *AM* meta-model, quite a few elements in an *AM* are sub-classes of another element. For example, *FormUnit* and *LogicU-*

nit are a type of *NavigationalUnit*. This implies that whatever property holds for a *NavigationalUnit* also holds for *FormUnit*. This fact has been used in the implementation of *Hydragen* by setting up the *AM* repository with RDF Schema Inferencing SAIL layer from Sesame. In particular, when a *FormUnit* or a *LogicUnit* is encountered in a *NavigationalUnit*, after handling elements specific to these units, the function to handle *NavigationalUnit* is called recursively. Same holds for *SubUnit*, *SetUnit* and *TourUnit* as well since they all refer to a *NavigationalUnit* which needs to be handled as part of current request. This works only because due to inference following query evaluates true for all sub-classes of *NavigationalUnit*:-

```
SELECT DISTINCT onload, input, attribute, formunit,
                relation,unit, scriptObject, webService
FROM {<reqNavUnit>} rdf:type {ams:NavigationalUnit};
                    [ams:onLoadQuery {onload}];
                    [ams:hasInput {input}];"
                    [ams:hasAttribute {attribute}];
                    [ams:hasFormUnit  {formunit}];
                    [ams:hasRelationship {relation}];
                    [ams:hasUnit {unit}];
                    [ams:hasScript {scriptObject}];
                    [ams:hasWebService {webService}]
```

In addition, it is useful to setup *DM* and *CM* repositories with inference in order to simplify some of the *AM* queries. For example, the following query:-

```
SELECT A
FROM {A} rdf:type {imdb_dm:Actor};
rdfs:label {X}
WHERE X Like "*$searchTerm$*"
```

and the following *DM*:-

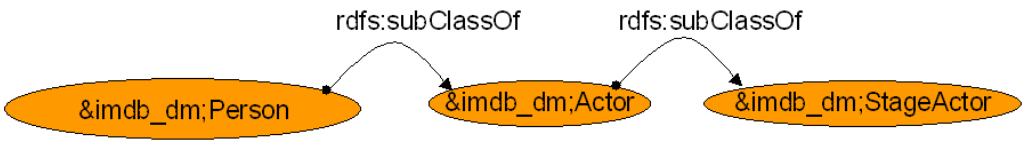

Figure 4.5: Inference

would normally only return the actors from the Domain model and not the stage actors. Although useful, care should be taken in using inference for Content repository since the performance penalty can be huge, during the initial setup of the database as well as when any modification is made to the database. Inference

SAIL would cause recalculation of derived statement whenever there is modification in the database. Sesame 1 supported "Custom inferencer", which allowed control over what rules must be used during inferencing so that user can do "data-dependent RDF inferencing", but this feature still needs to be implemented for Sesame 2.

How is multiple user access supported?

As specified in the Hera-S methodology description[24], the context data for an Hera-S application can be categorized into three types: session data, user data and global data.

Concurrency was one of the important requirements of the engine. This is realizable by providing support for session management and serialization of database access. The concurrency control at database level is realized with the help of Sesame (and choosing relevant SAIL layers within Sesame). Session Management deals with primarily two concerns:-

1. Identifying which request belongs to which Session.
2. Maintaining Session state across request.

HTTP is a session-less protocol. Each GET request is handled by

- Open a connection to the Web server
- Download the document
- Drop the connection

In the context of Hera-S, we intend to make a "stateful server", which implies that, most of the state information is stored as part of Context repository. The kind of state-information per session the Context repository can store includes:-

- SessionID
- Application specific context data such as contents of a shopping cart, flight selection etc.

The Hydragen engine does not handle any of this information directly. It all depends on the Context model and Application Model itself. The exact details of how to manage Session data are explained in Section 5.1.

How is debugging supported?

Debugging Web application is usually very difficult due to the dynamic nature and involvement of multiple components. In order to facilitate debugging of

HydragenCore and *HydragenWeb*, support for logging has been added to the application. For this purpose log4j library is used. It is possible to configure the level of logging by setting it in *log4j.properties* file of the *HydragenWeb* installation directory. The traces include information such as timing of events, data queries and Application model queries and flow of application handler etc.

4.3.3 Sesame related

As mentioned before, during development of *Hydragen* multiple versions of Sesame were used. Due to the fact that Sesame is still in a beta state, quite a few limitations were determined and resolved: some in Sesame and for some a workaround was found. Below some of such issues are described:-

Update Queries

Although there is support for SELECT and CONSTRUCT queries in SeRQL, it does not support update or delete queries. Since at *AM* level, updates and deletion are required in quite a few use cases and we do not want to rely on the programmable solution to support this (in order to simplify expression this within the *AM* framework), this was a serious limitation. In order to overcome this limitation, support for UPDATE and DELETE queries was added to *Hydragen*.

UPDATE and DELETE clauses were added with a specific syntax in order to facilitate the translation of these queries to SeRQL/Sesame constructs. For UPDATE queries, Hydragen allows support for at most one update expression in a query. The update expression must be encapsulated within *eval* function. Besides, within the update expression, the original node which needs to be updated must be placed between "*%*" signs. In order to update a node, first the current value and datatype of the node is retrieved, then the selected tuple with current value of node is deleted, update expression is evaluated and a new statement is created with the updated node. Note that Update query needs an RDF path to update, just providing the update expression for relevant node does not work. For example, the following update query which updates the number of times a user has viewed a Movie page:-

```
UPDATE {V} imdb_cm:hasNoOfViews {eval(%views%+1)}
FROM  {$U$} rdf:type {imdb_cm:RegisteredUser};
            imdb_cm:hasMovieViews {V} imdb_cm:hasNoOfViews {views};
            imdb_cm:hasViewsOfMovie {$M$}
```

is translated to a sequence of steps:-

1. Get current value of node to be updated

```
SELECT views,datatype(views)
FROM {$U$} rdf:type {imdb_cm:RegisteredUser};
     imdb_cm:hasMovieViews {V} imdb_cm:hasNoOfViews {views};
                               imdb_cm:hasViewsOfMovie {$M$}
```

2. Select and delete tuples with current value of the node.

```
delete_nodes =  CONSTRUCT {V} imdb_cm:hasNoOfViews {}
delete delete_nodes
```

3. Evaluate update expression.

```
VALUE = eval(%views%+1)
```

4. Add Tuple with new VALUE

```
CONSTRUCT {V} imdb_cm:hasNoOfViews {VALUE}
```

Creating new node

In *Hydragen*, application scenario such as registering of a new user are also handled within *AM*, but this requires creation of new RDF statements about username, password, age etc. The only way to identify a *subject* in RDF is by assigning a unique URI to it. This turned out to be little bit of challenge in the beginning, since in order to add a new node, one needs to determine a unique URI, which does not already exist in the RDF store. The solution was found in the handling of blank nodes in CONSTRUCT query by Sesame. Basically, Sesame creates a unique name (not URI) for a blank node in a Construct query with the blank node specified in empty curly brackets. The *Hydragen* engine is then able to convert the unique name into a URI in the *AMP* namespace, and this URI is used to create the new statements and add them to the RDF store. For example in the following *RegisterLogicUnit*:-

```
<am-metamodel:LogicUnit rdf:about="&imdb_am;RegisterLogicUnit">
 <am-metamodel:onLoadQuery rdf:resource="&imdb_am;newUserQuery"/>
 <am-metamodel:hasRelationship
    rdf:resource="&imdb_am;RegisterLoginRelationship"/>
</am-metamodel:LogicUnit>

<am-metamodel:UpdateQuery rdf:about="&imdb_am;newUserQuery">
 <am-metamodel:queryBody rdf:datatype="&xsd;string">
```

```
    Construct {} rdf:type {imdb_cm:RegisteredUser};
                 imdb_cm:hasName {"$userName$"};
                 imdb_cm:hasLoginName {"$loginName$"};
                 imdb_cm:hasLoginPassword {"$loginPassword$"};
                 imdb_cm:age {$userAge$};
                 imdb_cm:gender {"$userGender$"}
 <am-metamodel:queryType
    rdf:datatype="&xsd;string">SERQL</am-metamodel:queryType>
</am-metamodel:UpdateQuery>

<am-metamodel:Relationship rdf:about="&imdb_am;RegisterLoginRelationship">
 <am-metamodel:refersTo rdf:resource="&imdb_am;LoginUnit"/>
</am-metamodel:Relationship>
```

The specified *UpdateQuery* is a Construct query which creates a "new" node using curly brackets in the RDF path of Construct clause. This Construct query will result in RDF like shown below.

```
<rdf:Description rdf:nodeID=
   "http://wwwis.win.tue.nl/~hera/Hera-S/amp-schema#node12g10iogqx2">
<rdf:type rdf:resource=
   "http://wwwis.win.tue.nl/~hera/Hera-S/imdb_cm.owl#RegisteredUser"/>
<imdb_cm:hasName>New</imdb_cm:hasName>
<imdb_cm:hasLoginName>new</imdb_cm:hasLoginName>
<imdb_cm:age rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">26
   </imdb_cm:age>
<imdb_cm:gender>Male</imdb_cm:gender>
</rdf:Description>
```

Nested queries

Although Sesame 2.0 supports nested queries, they currently seem to have quite some performance issues. Nested queries could have been useful in implementation of *AMParser*, since for *SetUnit*, *AMParser* first collects all the results of a Set query and then for each result, execute a query to find the value of its attributes. A cleaner implementation would have been to handle the following *SetUnit* using Nested queries. Consider the following *NavigationalUnit* which is meant to create a menu of Favorite movies for a particular user:-

```
<am-metamodel:NavigationalUnit rdf:about="&imdb_am;MovieMenuNavigationUnit">
 <am-metamodel:hasUnit rdf:resource="&imdb_am;SearchRefUnit"/>
 <am-metamodel:hasSetUnit
```

```
    rdf:resource="&imdb_am;MovieUnitUserFavoritesSetUnit"/>
</am-metamodel:NavigationalUnit>

<am-metamodel:SetUnit rdf:about="&imdb_am;MovieUnitUserFavoritesSetUnit">
 <rdfs:comment rdf:datatype="&xsd;string">Favorites</rdfs:comment>
 <am-metamodel:refersTo rdf:resource="&imdb_am;MovieUnitUserFavoritesUnit"/>
 <am-metamodel:hasQuery rdf:datatype="&xsd;string">
    SELECT DISTINCT MovieFavElement
    FROM {$U$} imdb_cm:hasFavorite {MovieFavElement} ...
  </am-metamodel:hasQuery>
</am-metamodel:SetUnit>

<am-metamodel:NavigationalUnit
   rdf:about="&imdb_am;MovieUnitUserFavoritesUnit">
 <am-metamodel:hasAttribute  rdf:resource="&imdb_am;FavMovieName"/>
 <am-metamodel:hasRelationship
   rdf:resource="&imdb_am;MovieFavSetRelationship"/>
</am-metamodel:NavigationalUnit>

<am-metamodel:Attribute rdf:about="&imdb_am;FavMovieName">
 <rdfs:label rdf:datatype="&xsd;string">Movie Name</rdfs:label>
 <am-metamodel:hasQuery rdf:datatype="&xsd;string">
    SELECT T
    FROM {$MovieFavElement$} rdf:type {imdb_dm:Movie};
                             rdfs:label {T} ...
 </am-metamodel:hasQuery>
</am-metamodel:Attribute>

<am-metamodel:Relationship rdf:about="&imdb_am;MovieFavSetRelationship">
 <am-metamodel:hasAssignment>M=$MovieFavElement$</am-metamodel:hasAssignment>
 <am-metamodel:refersTo rdf:resource="&imdb_am;MovieMainUnit"/>
 <am-metamodel:sourceUnit rdf:resource="&imdb_am;MovieReferredSubUnit"/>
</am-metamodel:Relationship>
```

The above unit finds all the Favorite movies of a user and then for each movie, the name and the relationship are retrieved. This results in execution of *2n+1* queries where n is the number of Favorite movies retrieved for a user. Instead, if the same situation can be transformed to two nested queries:-

```
SELECT T
FROM {FavMovie} rdf:type {imdb_dm:Movie};
```

```
                rdfs:label {T}
 WHERE FavMovie IN {
    SELECT DISTINCT MovieFavElement
    FROM {$U$} imdb_cm:hasFavorite {MovieFavElement}
}

 SELECT FavMovie
 FROM {FavMovie} rdf:type {imdb_dm:Movie}
 WHERE FavMovie IN {
    SELECT DISTINCT MovieFavElement
    FROM {$U$} imdb_cm:hasFavorite {MovieFavElement}
}
```

But when these queries were tried out the Sesame, the resulting performance was worse than the *2n+1* queries. Hence, this transformation is not implemented in the current version of *Hydragen*.

4.3.4 Performance related

Dynamic generation of Web pages, bring with it the risk of performance penalty. Hence from the beginning, performance of *Hydragen* was a key requirement. Several design decision have been influenced by this requirement.

The choice that all generated *AMP* are kept in an internal data-structure before submitting is influenced by the fact that open/closing a transaction for every query consumes additional time, hence it is wise to open a transaction, completely handle a *NavigationalUnit*, submit the modifications and then close the transaction. Implementation was modified to follow this scheme.

AMParser uses SeRQL queries to extract the structure of *AM*. Initial implementation, executed around 500 queries for a *NavigationalUnit* of around 100 elements (including attributes, subunit, setunit etc). Hence, it took around 38 seconds to display the resulting page! The reason for this poor performance and the steps taken to optimize it are listed below:-

- Initially, the *AMP* was stored in the same RDF store as the *AM* although in a different context. But as discussed earlier, due to the fact that *AM* repository was setup with inference, it took quite some time to submit an *AMP* to the repository. Later, a separate in-memory repository without RDFS inference SAIL was setup for the *AMP*.
- Some queries were replaced by Sesame API calls. For example, originally a query was used to determine if the requested *NavigationalUnit* existed. This was replaced by a call to *Repository.hasStatement()*. This is supposed to be more efficient.

- SeRQL supports pattern matching on nodes found in a query by using *LIKE* clause. In case of *AMParser*, an exact match was needed in most cases hence using *LIKE* was costlier and irrelevant. This was resolved.
- As we saw in the *handleRootUnit* flow, in order to parse a *NavigationalUnit* all sub-elements need to be processed. A check was added for each element to be processed only if it exists.
- Maximum number of queries were being generated from the *SetUnit*, since for each retrieved element a *NavigationalUnit* must be parsed. In order to reduce this, the results of the Set query were stored and the referred *NavigationalUnit* was parsed once. During this parsing, for each result, attributes were queried and generated. This reduced the number of queries from *AM* quite a bit.
- Lastly, setting up *AM* on a server caused additional delay in parsing of *AM*. Hence, it was decided to use in-memory repository to store it instead.

After all these exercises, the response time for the *NavigationalUnit* in question went down from 14 sec to 3 sec. In this duration, around 200 queries were being run, generated *AMP* was being submitted and dumped to a file and finally XSLT transformation was generating the final HTML output. The current performance is considered sufficient for now due to the performance limitations of Sesame (like in case of nested queries).

In the end, all the application level bottlenecks have been removed from the implementation. Still there is potential for improvement by more optimization at the Sesame side. For example, improving the performance of nested queries would open opportunity for more efficient implementation.

4.3.5 Adaptation related

All the elements in *Hydragen* are dynamic in the sense that during execution of an application based on Hera-S models, contents of these models can be modified. This fact can be used to modify the Application Model and the Domain model on the fly to add new attributes to the application. In addition, the Aspect-Orientation feature of Hera-S requires that the queries within an Application model can be modified during execution.

4.3.6 Presentation Layer related

The *AMP* generated from *HydragenCore* is in RDF in XML serialization. This choice of serialization (instead of turtle, trix, n3 etc.) was made since it made the use of *XSLT* as transformation language easier. It must be noted that although XSLT does the job here, it is not the most effective transformation language for

RDF. This is due to the fact that there are multiple possible XML serializations for RDF. Moreover, XPath expressions, which are used extensively in XSLT rely on the XML tree structure while RDF is a graph. Though this made writing XSLT more cumbersome, no serious alternative could be found during the course of the project which would not need considerable effort. Some options that were considered include *Treehugger* and *RDFTwig*. These solutions make an attempt to add RDF specific Path expressions to XSLT. In addition to these options, a proposal was made to use XSLT extension points to allow SeRQL queries instead of XPath to retrieve RDF nodes. This option does look feasible but due to the scope of project, it was not followed up. These solutions should better be explored in future work.

4.4 Presentation Generation

Although the presentation layer of Hera-S methodology was not the core focus of this Master Thesis, still it was needed to prove the working of Hera-S flow and in order to develop a fully functional prototype. In fact, even before the design and implementation of *Hydragen* was started, a separate exercise was carried out to manually generate an *AMP* from an *AM* and also write an XSLT transformation to generate HTML result. Later on, the work from this exercise was used as starting point for implementation of Presentation Layer. Key point to note here is that presentation generation is not handled within the *HydragenCore* but instead is a part of *HydragenWeb*.

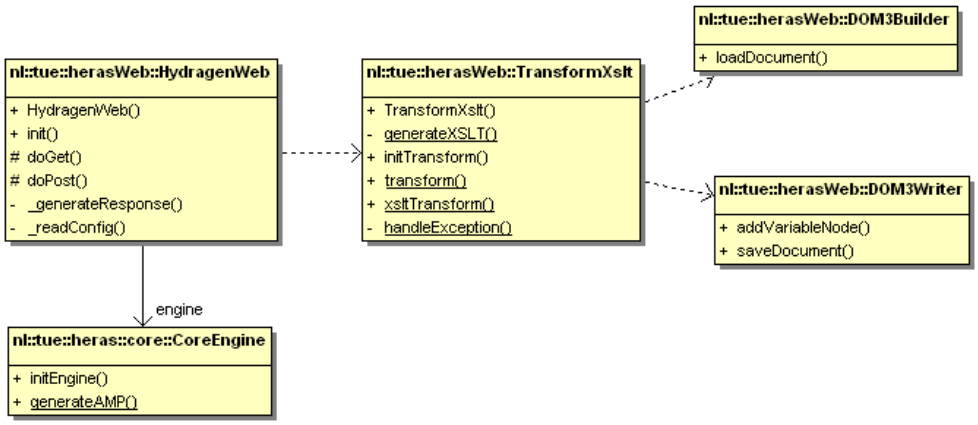

Figure 4.6: HydragenWeb Class Diagram

As is shown in 4.6, the *TransformXslt* class is responsible for-

- parsing the XSLT input
- creating an internal DOM3 structure
- defining some application specific XSLT variables
- generating the Presentation Model Instance
- transformation of *AMP* to HTML using the PM

In XSLT description, a separate template can be created for each *NavigationalUnit* for which an *AMP* is generated. XSLT uses XQuery to find specific nodes within the RDF/XML and generate HTML code using the contents of these nodes. Since, there is no unique XML serialization of RDF, *xsl:variable* must be used extensively to reach a specific node in smaller steps and then its value must be extracted. For example, following XSLT extract shows how to define a template to generate HTML only for a *LoginUnit*.

```
<xsl:template match=
        "/node()[rdf:Description[ends-with(@rdf:about,'LoginUnit')]]">
    <xsl:call-template name="LoginTemplate"/>
</xsl:template>

<xsl:template name="LoginTemplate">
    <HTML>
        <HEAD>
            <META http-equiv="Content-Script-Type"
                  content="text/javascript"/>
        </HEAD>
        <BODY style="...">
        <div id="header">...
        </div>
        <div id="body">...
        </div>
        </BODY>
    </HTML>
</xsl:template>
```

In order to determine, the value of an element (*LoginUnit* Title and the corresponding *FormUnit* query in this case) generated in the *AMP* following XSLT code can be used.

```
<!-- Show LoginUnit Title -->
<xsl:variable name="thisTitle">
```

```
<xsl:value-of select=
      "//rdf:Description[ends-with(@rdf:about,'LoginUnit')]/
         amp:hasAttribute/text()[ends-with(.,'LoginLabel')]"
/>
</xsl:variable>

<xsl:variable name="thisForm">
<xsl:value-of select=
     "//rdf:Description[ends-with(@rdf:about,'LoginUnit')]/
        amp:hasFormUnit/@rdf:resource[ends-with(.,'LoginForm')]"
/>
</xsl:variable>

<xsl:variable name="thisFormQuery">
<xsl:value-of select=
     "//rdf:Description[ends-with(@rdf:about,string($thisForm))]/
        amp:hasQuery/text()"
/>
</xsl:variable>
```

In the above XSLT code, XPath expression is used to extract the complete URI of a RDF object, which ends with 'LoginForm' and stored as an XSLT variable. After all elements have been extracted and stored in XSLT variables, these variables can be used within HTML to generate presentation elements.

```
<table border="0" cellpadding="5" cellspacing="0">
<form method="post" action="{$thisFormRefersTo}">
<tr><td>ID:</td>
    <td><input type="text" size="20"
             name="{$thisLoginUserNameInputBindingVar}"
             value=""/>
    </td>
</tr>
<tr><td>Password:</td>
    <td><input type="password" size="20"
              name="{$thisLoginPasswordInputBindingVar}"/>
    </td>
    <td><input type="submit"
              value="{$thisButtonText}"/>
    </td>
    <td>|</td>
```

```
    <td><A href="{$thisRefersTo}"
        onMouseover="register.src='./images/Registerover.png'"
        onMouseout="register.src='./images/Registerup.png'">
         <img src="./images/Registerup.png"
              border="0"
              name="register"
              padding-left="50px"/>
        </A>
   </td>
</tr>
</form>
</table>
```

4.5 IMDB example

As a demonstration of the capabilities of the Hydragen, an extensive Web application was built using the Hera-S methodology. This example application was motivated by the IMDB Web site which provides information regarding movies, actors etc. A considerably large sub-set of movie database was used for this application. Moreover all the relevant models, namely *AM*, *DM* and *CM* were constructed to create a full-fledged application.

The detailed AM used for this application can be found in B.2. In brief, it support the following flow:-

- Display login screen, with option to enter user credentials and submit the login form or Register a new user.
- Register form for new user which takes user back to Login screen after registration.
- If login fails, user is sent back to Login form with a login failure message. After successful logging in, user is presented with a personalized welcome message and a Search Form.
- Search Form allows user to input a key word and select whether he/she wants to search for movie or actor.
- Depending on choice of user, he/she is shown a Movie result page or an Actor result page.
- User can click on the entity he/she was searching on and he is presented with a page about movie or actor as the case be.

- A movie unit shows various attributes of the movie/actor, some pictures of the movie and a table of actors/actresses who were part of the movie cast. Besides, an Ajax based side menu is displayed with list of Top 10 movies and Favorite movies as indicated by the User. The page also displays a button to add current displayed movie to users favorites. Similar structure is also followed for the Actor unit.

Figure 4.7 is a capture of an *MovieUnit* as created by Hydragen.

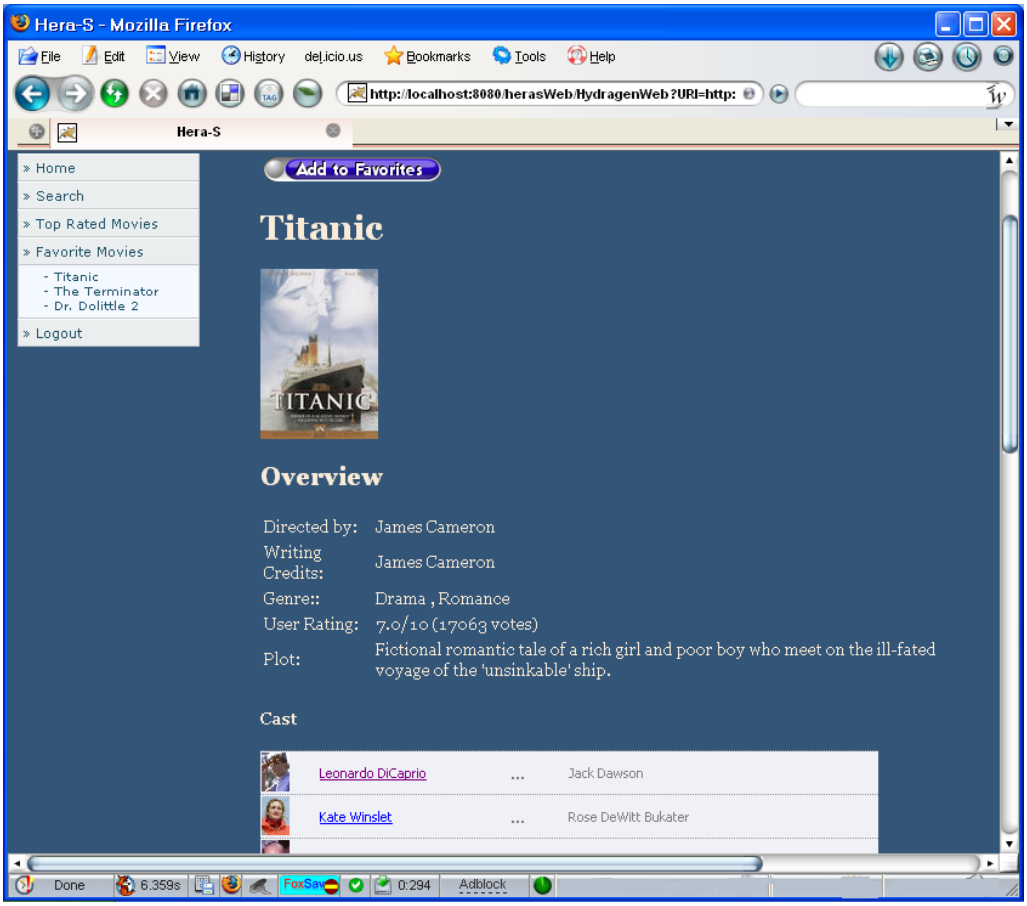

Figure 4.7: IMDB - Hera-S demo

Chapter 5

Hydragen: Usage

Application of *Hera-S* methodology sub-divides the task of creating a Web application into smaller manageable steps motivated by separation of concerns. This section describes the steps needed to develop an example application based on IMDB movie database. Besides, some of caveats that need to be avoided for are mentioned as well.

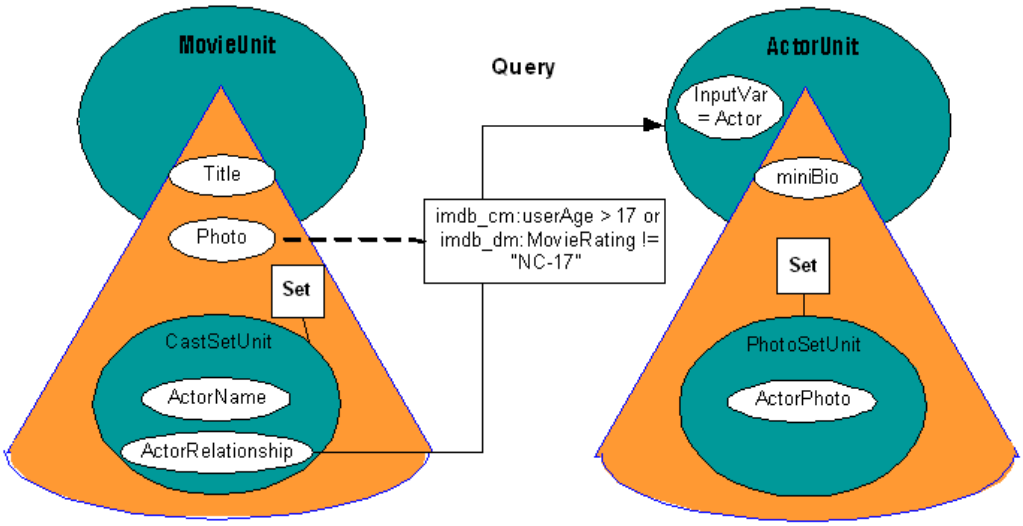

Figure 5.1: IMDB Application Model

5.1 Creating new application

1. *Create DM*: Since *Hera-S* targets data intensive application, the starting point for any Web application developed based on *Hera-S* methodology is

the domain model. The domain model captures the relationship between elements in the data over which the application needs to operate. For example, the figure 5.2 shows relation between various properties of a Movie that are captured in the data content.

2. *Create CM*: While *DM* deals with modeling of data content, context model captures relations between information related to personalization and adaptation of content delivery. The main difference between *DM* and *CM* is that context data is typically under direct control of *Hera-S*. For example, the *UpdateQuery* in the *AM* is mostly expected to deal with elements from *CM*. This includes:-

   - *User data* i.e. personal information for the purpose of maintaining a user profile in the application. For e.g. user name, login name, password, favorites, personal rating etc can be modeled as part of the Context model. Figure 5.3 shows the Context model for user specific data such as favorite movies of a *RegisteredUser*.
   - *Session data* i.e. information that is valid only till a session is alive. As soon a session expires (by default after half an hour of inactivity), the related data is deleted. Examples of such data include contents of a shopping cart if there is no user logged in, or choice of travel destinations saved for later etc. Since there is no user to which this information can be related, the current *sessionID* is used as an anchor. The *AM* should store all session specific data in a Session specific context within the Data repository. *HydragenWeb* passes the current *SessionID* to *HydragenCore* as a parameter so that the *AM* can identify which *SessionID* to store the *Session data* in. The value of *SessionID* is used to construct a context URI which is used the value of a predefined *AM* variable - *sessionID*. Since before any query, all variables are replaced by their values, an Construct/Select query which contain this *sessionID* variable is also "instantiated". An example of such a Session query is following:-

     ```
     Construct {} rdf:type {imdb_cm:ShoppingCart};
                  imdb_cm:hasItem {"$item$"}
                  imdb_cm:hasCount {"$Number$"}
     To Context $sessionID$
     ```

     Note that SeRQL does not support "TO CONTEXT" construct. It is interpreted by *Hydragen*, and all the statements generated by the Construct query are added to the specified context URI. If there is not context existing with the given URI, it is automatically created.

     It is the responsibility of the *AM* creator, that *Session data* is update when needed. For each request, *HydragenWeb* checks if the session

corresponding to the request is still valid. In case, the current session is expired *HydragenWeb* informs *HydragenCore* by issuing a request to a predefined *SessionExpiredLogicUnit*. Again, it is the responsibility of the *AM* creator to provide for such a logic unit whose main purpose is to delete all Session related data and potentially redirect the user to "Home" page of the application. The delete query which removes all Session data corresponding to a specific session will look as follows:-

```
Delete * from context $sessionID$
```

- *Global data* i.e. the information that is relevant at an application level. Examples of global data include "most visited link" or "recommendation for related items".

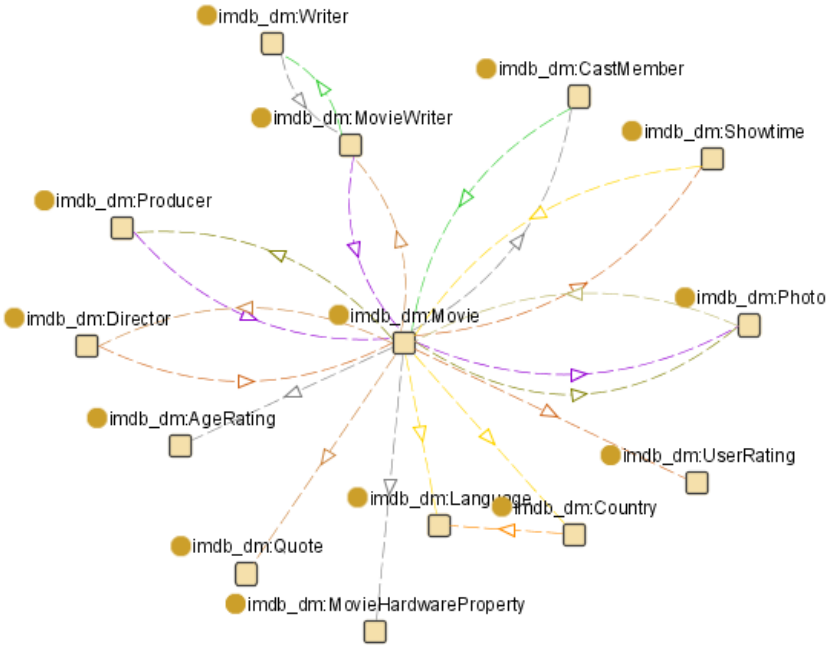

Figure 5.2: IMDB Domain Model

Note all these models can be developed in RDF or OWL using tools such as Protege, which is an open source ontology editor.

3. *Create AM*: The core functional and navigational behavior of the Web Application is described in the *Application Model*. Its wise to use the *Application Meta-model* i.e. the OWL schema for *AM* as the starting point

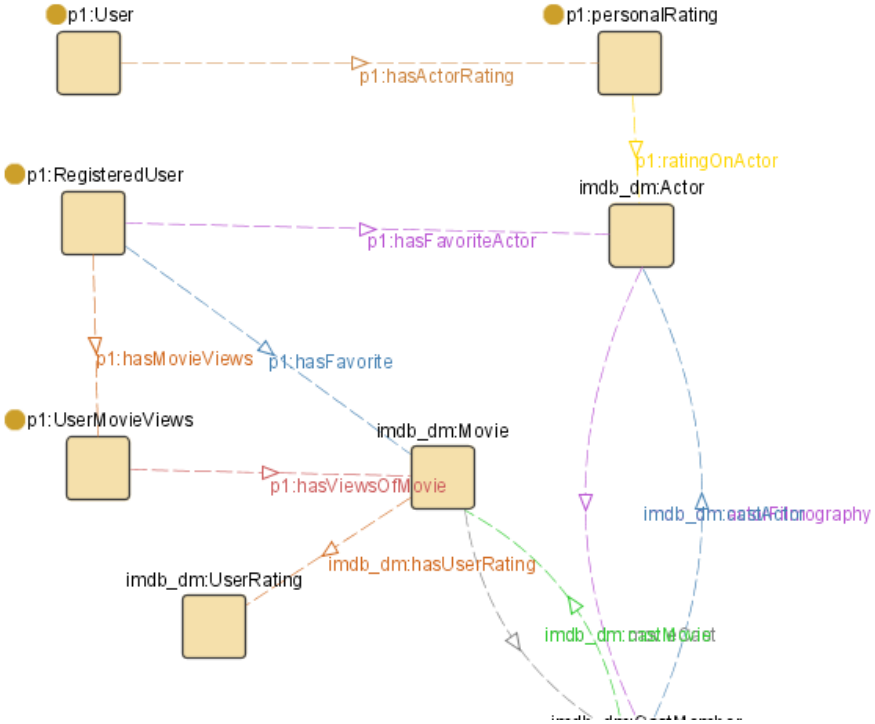

Figure 5.3: IMDB Context Model

for development of an *AM*. Using tool like Protege to create instance of *AM* elements such as *NavigationalUnit* and their relevant attributes makes defining of these navigational structure extremely manageable. For previous versions of *Hera* a GUI Builder is available but it has yet not been adopted to work with *Hera-S* and this would perhaps be picked up in one of future Hera projects. In general, the fact that *Hera-S* relies on standard formats such as RDF(S) and OWL and that there are several tools that allow user to create content and visualize it in these formats, makes it less relevant to develop a *Hera-S* specific editor.

There are certain caveats that the developer must be aware of while creating an *AM*. These are explained in Section 5.2.

4. *Create PM*: Next step is to describe how each *NavigationalUnit* would be presented. The current *HydragenWeb* setup only allows XSLT description for the *PM* and it expect HTML output but in principle other output formats such as SMIL, WML etc can also be supported with minor extensions.

5. *Setup Web Server*: Both *HydragenWeb* and Sesame need a Java Servlet Container such as Apache Tomcat 5.5 to host them. Once Apache Server

has been installed, Sesame (version 2.0 beta5) server and client need to be deployed as instructed in User Manual for Sesame[7]. *HydragenWeb* Web application can be installed on a different machine with a separate installation of Web Server.

6. *Setup Sesame Repository*: Once the Domain and Context models have been created, a persistent Sesame RDF repository should be setup for the RDF models and Data content. The RDF repository for Application data, *DM* and *CM* must be setup as a remote native repository. Then, the content, *DM* and *CM* must be loaded into the server using the Sesame Client interface or programmatically. It is important to setup the Repository as persistent to preserve the data and context information valid even if the Server must be brought down for upgrade etc. Native Store as supported by Sesame 2.0 can be exploited to provide persistence of data.

7. *Create Hydragen Configuration File*: *Hydragen Configuration File* (example_server.xml) specifies the location of various models in XML format. The configurable items that can be set include:-

   - *application name* : The application root node allows the specification of a unique name for the application.
   - *amp_context* : URI of Sesame Repository context in which generated *AMP* is stored.
   - *app_server* : URI of the *HydragenWeb* Servlet. The value here is added to the *PM* (XSLT file) as an XSLT variable.
   - *pmpath* : Absolute or relative (to app_server path) path to the *Presentation Model*.
   - *amspath* : Absolute or relative path to the *AM meta-model*.
   - *ampath* : Absolute or relative path to the Application Model.
   - *Server setting* : Settings regarding location of Sesame server and names of repository for *DM*,*CM* and *Content* can be set separately using *dm_server_path*, *cm_server_path* and *content_server_path* elements. For each element following two properties need to set:-
     - *server_path* : URI of Sesame RDF server
     - *repository_id* : Unique name of Repository ID given during setup.

8. *Update Hydragen Deployment Descriptor*: Deployment Descriptor file (WEB-INF/web.xml) for *HydragenWeb* web application defines an initialization parameter called *configPath* which defines the path to Hydragen Configuration File. The value of this parameter must be set to the location (absolute or relative to app_server) of the *example_server.xml*. After updating web.xml, *HydragenWeb* must be reloaded.

After these steps, the *HydragenWeb* based application is ready to be used. Just browse to the app_server URI using your browser in order to visit the root page of your application.

5.2 Usage Caveats

There are certain caveats that the user must look out for while using *Hydragen*. These are listed below:-

- *AM* related
  - *Use of variables* : Since variables in *AM* have a global scope, they stay valid across multiple requests. This is useful in most of cases since variables such as "logged-in-user" is needed to be defined throughout the application. But, this also implies that the *AM* Designer must be careful with the use of variable names lest some variable defined in another *NavigationalUnit* is unintentionally overwritten. Also note that variables do not need to be explicitly defined, they are also created as part of query processing. The Select query variable is used as a binding variable for the result of the query.

    The *AM meta-model* can be extended to provide more flexible support for variables such as definition of scope of validity etc.
  - *Single attribute Queries* : The data retrieval queries in *AM* are all expected to extract a single attribute e.g. following query is illegal :-

```
SELECT DISTINCT Name, Photo, Cast
FROM {$A$} rdfs:label {Name};
     rdf:type {imdb_dm:Actor};
     [imdb_dm:hasMainPhoto {} imdb_dm:photoURL {Photo}];
     imdb_dm:actorFilmography {F} imdb_dm:castMovie {$M$},
                              {F} imdb_dm:castCharacter {Cast}
```

    Here a single SeRQL query allows retrieval of Name, Photograph URI and role of an actor in a given movie. But, since an Attribute definition with *AM* allows more information to be expressed per Attribute such as media type and a recognizable RDF label such queries are not valid in *AM*. In case such queries are used in *AM*, a comma separated concatenation would be returned as result.
  - *Namespaces in queries* : The queries specified in *AM* should be complete SeRQL queries including the "USING NAMESPACE" clause. In future versions of *AM meta-model*, it should be advisable to allow these definitions at a global level in order to keep the *AM* readable.

  - *Special Characters* : XML has as a set of special character which need to be encoded before using them as value of any entity. These include characters such as '⟨', '⟩' and '&' which should be written using the corresponding named entity encoding such as '<', '>' and '&'. Since RDF and XSLT must comply to XML Schema this requirement holds for the models and the RDF data as well. This needs special mention since *AM* models contain SeRQL queries which use these characters (especially '⟨' and '⟩') for namespace/URI definitions.
  - *Attributes and Relationships* : Current version of *AM* does not allow *Relationship* to be attached to an *Attribute* directly. This implies in order attach a link to an *Attribute*, a dummy *SubUnit* must be created which would contain the *Attribute* and a *Relationship* to the referred *NavigationalUnit*. *AM* can potentially be simplified to allow this "shortcut".

- *PM* related

  - *Frames* : XSLT transformation on *AMP* in general are expected to generate HTML to stdout which can then be passed to *HttpServletResponse*. In case of frames, XSLT "result-document" clause can be used to generated multiple HTML output, one per frame and the top level HTML document must be sent to stdout. At *AM* level as well, each frame must be modeled as a *SubUnit* of an overall main unit in order to capture the structure. This information can then be utilized by *PM* to place each *SubUnit* in its intended frame. Note here that *AM* does not specify frames, only frame-like behavior.

    Although, frames used to be popular to divide a unit of display in multiple regions and only updating the required regions, similar functionality can be achieved using CSS, Ajax and using div sections in HTML. In the IMDB example created using *Hydragen*, an Ajax based menu is used for *MovieMenuUnit* with the users Favorite movies, link to Search page etc was generated by generating a temporary XML menu structure based on the Favorite movies Set in *AMP* for *MovieMenuUnit* and using Ajax script to parse and display the menu.

5.3 Extension possibilities

One of the requirements for *Hydragen* was that it is extensible. The API for *HydragenCore* was defined keeping this in mind. The *nl.tue.heras.core.IEngine* and the *nl.tue.heras.backend.IBackend* interfaces were defined to provide extension points for alternative implementation of the engine.

In this context, *Hera-S* support for adaptation using Aspect-oriented concept (*Hera-S* [24]) is worth mentioning. Briefly, *Hera-S* allows dynamic transformation of *AM* queries to express a "cross-cutting" adaptation "concern" applicable to a certain set of *AM* elements. The definition of transformation is called *Advice* while *Pointcut* selects element on which the *Advice* is applicable. The support for Aspect-oriented concept was not part of this master project, but it was a requirement that *Hydragen* must be built in a way such that these extensions are feasible.

From a *Hydragen* perspective, support for advice/pointcut pairs primarily boils down to being able to extract specific *AM* elements and transforming the queries attached to these elements (adding conditions, or nested queries etc). The *HydragenCore* API provides access to a "handle" to *Application Model* which can be used to extract elements containing queries from *AM* by using API calls such as *Collection⟨ String ⟩ handleQuery(DbHandle handle, String query)*from the *nl.tue.heras.DbManager* class. The same function can be used to replace the current SeRQL query with transformed version using the UPDATE query syntax as explained in Section 4.3. Note the modification are only effective after a call to *submitStatements()* function. An alternative solution would be to use *AMhandle* to get access to *AM* repository and use Sesame Repository API to do the needed transformations . This works since the current *AM* is parsed on every Unit request. Thus, it is easy to extend the functionality of *Hydragen* using the provided API.

Similarly, applying *Hera-S* for development of a *Workflow Management System* is also feasible. The basic elements of a WfMS are *States, Tasks, Guards* and *Triggers*. It is possible to model all these elements in RDF as part of Domain model and Context model. The user and state dependent views of tasks that and possible transitions between them can be modeled using *Application Model*. The user specific information like current list of tasks assigned to him/her can be described in the Context model. The "business rules" that govern the valid transitions between different states need to modeled outside *HydragenCore* potentially using Prolog or OWL specification. Given current state, trigger, and user input it should be able to generate information about what should be next state and who should be the owner of the task that needs to be done in that state. This information can be sent as a request for required *NavigationalUnit* with corresponding parameter values like state, user name etc to *HydragenCore*.

Chapter 6

Hydragen: Analysis

There are several criterion on the basis of which *Hydragen* and the *Hera-S* methodology can be evaluated. These include ease of use, maintainability, extendibility, debug-ability etc. Most of these non-functional requirements are not easy to measure but the design decisions that were presented in last chapter address these aspects of the application as well. In addition to these, development time for a new application and time taken to update an existing application, are also indication of usefulness of *Hera-S* methodology. In case of a Web application which does not use modeling and automatic generation approach, all pages have to be manually written and maintained. Simple change in data content or change in presentation template can result in quite some work. Modeling based design methodologies and *Hera-S* in particular, saves all this development time and effort. Though no separate measurements were done in this respect, during the development of IMDB example, the benefits were immediately evident.

In the following sections, we focus on one important and concretely measurable evaluation criteria namely performance.

6.1 Performance

Performance of a Web application based on *Hydragen* is typically measured by the response time that a User experiences after he/she issues a request for a link generated by *Hydragen*. This is because *Hydragen* is dynamic and thus runs in the background all of the time.

There are multiple factors which influence the response time of *Hydragen* can be evaluated. Here we will analyze the performance by looking at the algorithmic complexity of handling a request and also actual response time data retrieved from example application.

6.1.1 Algorithm Complexity

In order to evaluate the algorithmic complexity of *Hydragen*, we determine the number of queries (both data and application queries) needed to be run to generate an *AMP* for each request. The number of executed queries are used as unit of performance since they are the most determining factor in handling of a unit. For the purpose of complexity evaluation, a typical example with the request unit that has at least one instance of each type of navigation element was used. This is not the general case. The number of distinct navigation element are denoted by n while the number of results retrieved for a *SetUnit* are denoted by m. Following the algorithm as depicted in figure 4.4 we observe that although for *SetUnit* and *TourUnit*, a *NavigationalUnit* must be handled once, for each result a distinct data query must be run to retrieve the value of attribute and relationship corresponding to that result(See Section 4.1). Based on this, we get $O(n + m)$ as an asymptotic upper bound for the algorithm.

In order to give an estimate, a navigation unit with exactly one element of each type, and a *SetUnit* which results in k elements and refers to another unit with one element of each type will execute - $2*k+2*17$ queries. The addition factor comes from the fixed number of AM and data queries that need to be handled per *NavigationalUnit* if all elements are present. Since the *NavigationalUnit* contains a *SetUnit* as well, these queries need to run twice, hence the multiplication factor. The number of queries executed to generate *AMP* for attribute and relationship of each element of *SetUnit* contribute to the $2k$ factor.

Navigational Unit	Nr of AM Statements	Nr of Data Queries in AM	Total Nr of Executed Queries	AM->AMP	AMP->HTML	Total time	Time per query	Average time per query
MovieUnit	106	28	151	2.514	0.600	3.114	0.016649	0.018388
			163	2.734	0.410	3.144	0.016773	
			163	2.574	0.651	3.225	0.015791	
			167	3.074	0.751	3.825	0.018407	
			167	2.804	0.701	3.505	0.01679	
			171	3.886	0.460	4.346	0.022725	
			219	4.726	0.732	5.458	0.02158	
ActorUnit	54	15	66	0.791	0.131	0.922	0.011985	0.014788
			166	2.403	0.471	2.874	0.014476	
			222	3.154	1.733	4.887	0.014207	
			294	4.747	2.313	7.060	0.016146	
			354	5.638	3.024	8.662	0.015927	
			694	11.096	12.458	23.554	0.015988	
LoginUnit	25	1	13	0.311	0.250	0.561	0.023923	0.023923
RegisterLogicUnit	5	1	19	1.962	0.181	2.143	0.103263	0.103263
ActorSearchResultUnit	11	2	24	0.390	0.091	0.481	0.01625	0.01625
AddToFavoritesUnit	4	1	379	5.428	3.255	8.683	0.014322	0.014322

Figure 6.1: Performance Data

6.1.2 Statistical Analysis

The example IMDB application was used to evaluate performance of *Hydragen*. In order to avoid network communication overhead, the Sesame Data Server and

Hydragen Application were installed on the same machine and also accessed from the same machine. A standard PC configuration with 1.73 GHz CPU and 1 GB RAM, running Windows XP SP2 was used for purpose of this test setup. The Application Model (Appendix B.2) contains 361 RDF statements with around 20 *Navigational/LogicUnit*'s and 46 data queries distributed over various units with as high as 28 data queries in *MovieUnit*. The IMDB data itself contained more than 240000 RDF statements with around 20000 movie titles, 2500 actor names and related information. Note that in the data set, quite a few movies/actors dont have all attributes (like hasMainPhoto, movieYear, actorFilmography etc) filled in. In particular, the number of movies with completed related information are 100; similarly for actor around 600 have sufficient attributes. Note all this data is preloaded to a Sesame server, except for Application Model, which is only loaded when the *HydragenWeb* application is initialized.

The data collected during this experimental setup is shown in Table 6.1. For this purpose, the following steps were carried out:-

1. Open the login page of application (*LoginUnit*)
2. Register new user (*RegisterLogicUnit*)
3. Search Actor (*ActorSearchResultUnit*)
4. Select Actor from Result (*ActorUnit*)
5. Add a Actor to User Favorites (*AddToFavoritesUnit*)
6. Select Movie from Top 10 list (*MovieUnit*)

For each of step, the following data was collected:-

- Number of statements in the *AM* for requested *NavigationalUnit*
- Number of Data queries in the requested *NavigationalUnit*
- Total number of queries executed to process the requested *NavigationalUnit*. This includes the parse queries, which are executed to determine the structure of *AM* as well as the Data queries that instantiate the *AM* to generate AMP. Note that the number of queries in the table vary based on the content, mostly because of the number elements in a *SetUnit*.
- Time taken from when the user clicked on a link to the complete generation of an AMP.
- Time taken to transform generated *AMP* to HTML and send the response to user.

Based on this data, the following statistics were derived:-

- Total time taken to handle a request.
- Ratio of time taken to generate *AMP* wrt the number of executed queries in order to normalize the execution times. This would be called *TPQ* (time per query).
- Average value of *TPQ* for each type of *NavigationalUnit*.

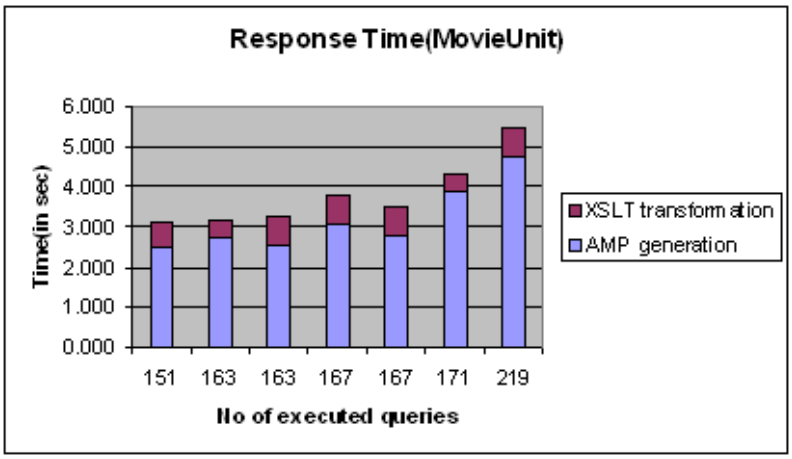

Figure 6.2: Response Time: Movie Unit

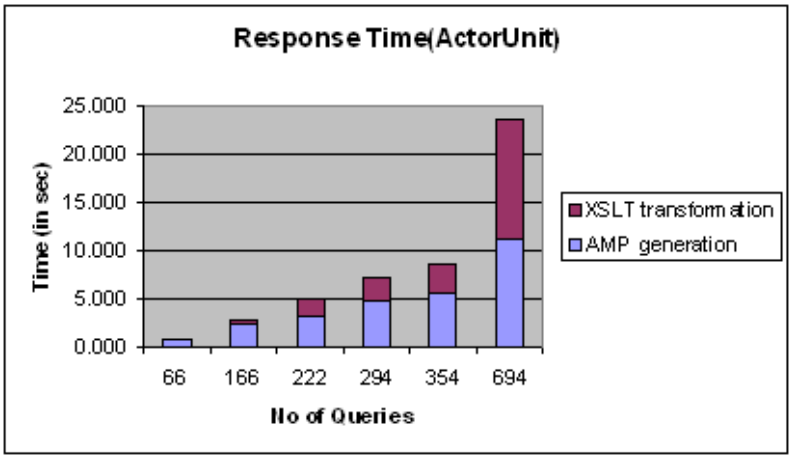

Figure 6.3: Response Time: Actor Unit

As is expected and can seen in figure 6.2 and figure 6.3, the response time for a request increases with number of queries that are executed to handle a *NavigationalUnit*. The number of executed queries is itself dependent on the number of *AM* statements and the number of results retrieved as part of *SetUnit* query. For example the last entry in figure 6.3, corresponds to generation of unit

for Actor "Tom Arnold" who acted in around 81 movies according to the movie dataset. The huge number of results returned by *SetUnit* query explain the high response time. The time taken for XSLT transformation from *AMP* to HTML is mainly dependent on the size of generated AMP.

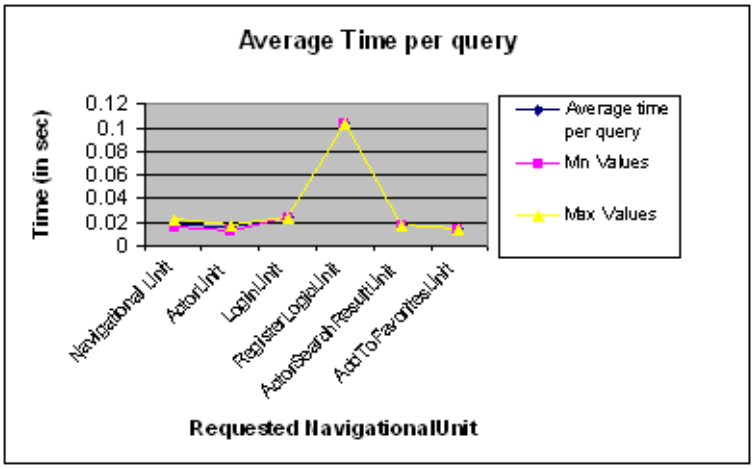

Figure 6.4: Average Time per Query

Figure 6.4, how average time to handle a query varies from Unit to unit depending on the type of queries in a unit. As can be observed, the average query handling time for *RegisterUnit* is higher than the rest of the units since it is dominated by a bigger Construct query while rest of the unit have only Select queries or simple update query. The Construct query creates all RDF statements corresponding to a User's profile (name, age, login, password etc). The addition of these RDF statements back to data repository takes extra time (in addition to the processing of query).

In addition to the average, the minimum and maximum time for handling a query for each type of Unit is shown in the graph.

Chapter 7

Conclusion and Suggestions for Future Work

Within the context of *Hera* project, we implemented *Hydragen* to support *Hera-S* design methodology. The initial requirements for *Hydragen* have been met as good as possible (except for performance problems due to Sesame) and an example application was developed to demonstrate the capabilities of the methodology as well as implementation.

For the methodology perspective, Hera-S was the logical next step in Hera development. By utilizing the power of standard Semantic languages such as RDF(S) and expressiveness of SeRQL, *Hera-S* capitalizes on the availability of multitude of tools that support model development, verification and visualization. If needed, SPARQL can be easily incorporated in Hera-S. In particular, the availability of Application Metamodel in RDFS makes development of an *AM* a simple task of instantiating the various elements with specific values.

During development of *Hydragen*, it became clear that although Sesame 2.0 provided some promising features; there were still some issues since that it was in alpha/beta stage. Having started with using Sesame 2.0 alpha4, quite a few of these issues were resolved in cooperation with Aduna. The work on *Hydragen* helped making Sesame better. The final *Hydragen* implementation is using Sesame 2.0 beta5 release, although still some performance related issues are open.

In compliance with the philosophy of layered development of *Hera-S*, *Hydragen* was also implemented modularly. The *HydragenCore* was implemented first as a standalone tool and modified to a library for use with *HydragenWeb*. This approach motivated the development of a clean API for *HydragenCore* which can find use in development of other Web applications.

7.1 Suggestion for future work

The present *Hydragen* prototype is a working application, which performs within the given requirements; there is however still room for improvements. In the field of performance:

- As discussed in Section 4.3, *Hydragen* can benefit from support of efficient Nested queries by Sesame. Once this is in place, the algorithmic complexity can drop from *O(n+m)* to *O(n)*, since then handling of *SetUnit* would not depend on number of results for *SetUnit* query. Note that the gain in terms of response time wont be linear since the execution time of a nested query is expected to more than that of a non-nested query.

- Currently, if a part of *NavigationalUnit* needs to be updated while keeping the rest same, it requires complete regeneration of *AMP*. E.g. in case of *AddToFavorites* unit after processing of the update query, application just needs to return to the currently displayed page (*MovieUnit* page in this case), with an updated Favorites menu. In this case, it would have sufficed to update the Favorites *SubUnit* only. An approach that allows supporting this enhancement could be to store each *SubUnit* in a different context of *AMP* store. Then the statements from only a specific context can be replaced if the requested unit is same as current unit. Of course, then the *SubUnit* to be updated must be encoded in the request as well. Though this is not currently supported in *Hydragen*, with some effort the implementation can be made smarter and hence result in improved performance.

In addition, quite a few shortcuts can be supported at the *AM* level which would simplify the work of an *AM* Designer. These include:-

- Global specification of namespace definitions for data queries: Currently, every data query in *AM* needs to specify namespace definitions separately, making the *AM* unreadable. This can be simplified by making global definitions and allowing Hydragen to complete the queries before executing them.

- Support for relationship in Attributes: Currently, the rule is that each *NavigationalUnit* can have a unique relationship. This means in order to attach a link to an *Attribute*, one needs to create a *SubUnit* and place a *Relationship* in the *NavigationalUnit* referred by this *SubUnit*. Although a clean approach, this is unnecessarily complicated and can be solved by supporting *Relationship* for *Attributes*.

- Abbreviated queries: In quite a few situations a query is needed only to retrieve the value of an attribute without an condition to be specified. It is

possible to simplify the syntax of query to be written just using RDF path which contains one SeRQL variable. E.g. Instead of

```
Select x from {x} rdf:type imdb_am:Person using namespace...
```

it should be possible to write

```
{x} rdf:type imdb_am:Person
```

Hence, making *AM* more readable.

Finally, the Presentation Layer implemented in *Hydragen* is minimal. It does not provide any adaptation possibilities apart from these expressed in XSLT itself. Since the focus of the project was to develop the *HydragenCore* functionality, little effort was spent on extending support for Presentation Model. This part can benefit from already proven possibility of integrating Hera and AMACONT[23].

Bibliography

[1] Jena. http://jena.sourceforge.net/. [cited at p. 5]

[2] Protege. http://protege.stanford.edu/. [cited at p. 5]

[3] Racer. http://www.racer-systems.com/. [cited at p. 5]

[4] Redland. http://librdf.org/. [cited at p. 6]

[5] Saxon. http://saxon.sourceforge.net. [cited at p. 41]

[6] Sesame. http://www.openrdf.com/. [cited at p. 5]

[7] *Sesame-User Manual*. http://www.openrdf.com/doc/sesame2/users/index.html. [cited at p. 63]

[8] W3c. http://www.w3c.org/. [cited at p. 5]

[9] Xerces2. http://xerces.apache.org/xerces-j. [cited at p. 41]

[10] Parsia B. and Sirin E. Pellet: An owl dl reasoner. *International Semantic Web Conference*, 2004. [cited at p. 5, 41]

[11] Stefano Ceri, Piero Fraternali, and Stefano Paraboschi. Data-driven, one-to-one web site generation for data-intensive applications. In *The VLDB Journal*, pages 615–626, 1999. [cited at p. 6]

[12] Rossi G.; Pastor O.; Schwabe D. and Olsina L. *Web Engineering: Modeling and Implementing Web Applications*, volume 12 of *Human-Computer Interaction Series*. 2007. [cited at p. 18, 28]

[13] Schwabe D. and Rossi G. *An Object Oriented Approach to Web-Based Application Design*, volume 4 of *Theory and Practice of Object Systems*. Wiley and Sons, New York, 1998. [cited at p. 7]

[14] O. De Troyer and S. Casteleyn. Modeling complex processes for web applications using wsdm. In *Proceedings of the Third International Workshop on Web-Oriented Software Technologies*, 2003. [cited at p. 8]

[15] Lei Y.; Motta E. and Motta J. *Modelling Data-Intensive Web Sites with OntoWeaver*, volume 2 of *International Workshop on Web Information Systems Modeling*. 2004. [cited at p. 7]

[16] Balasubramanian P.; Isakowitz T.; Stohr E.A. Rmm: A methodology for structured hypermedia design. In *Communications of the ACM*, pages 33–44, 1995. [cited at p. 6, 11]

[17] Jaime Gómez and Cristina Cachero. Oo-h method: extending uml to model web interfaces. pages 144–173, 2003. [cited at p. 8]

[18] Berners-Lee T.; Hendler J. and Lassila O. The semantic web. *Scientific American*, 2001. [cited at p. 5]

[19] Stuckenschmidt H.; Vdovjak R.; Broekstra J. and Houben G.J. *Towards distributed processing of RDF path queries*, volume 2 of *Int. J. Web Engineering and Technology*. 2005. [cited at p. 30, 44]

[20] Hinz M. and Fiala Z. *AMACONT: A System Architecture for Adaptive Multimedia Web Applications*. Berlin, Germany, October 2004. [cited at p. 18, 29]

[21] Ceri S.; Fratenali P. and Bongio A. *Web Modelling Language (WebML): a modelling language for designing Web sites*. Amsterdam, May 2000. [cited at p. 8]

[22] Balpreet Singh. *Hydragen: User Manual*. [cited at p. 30]

[23] Houben G.; Fiala Z.; Sluijs K. van der and Hinz M. *Building Self-Managing Web Information Systems from Generic Components*. June 2005. [cited at p. 29, 75]

[24] Sluijs K. van der; Houben G. J.; Broekstra J. and Casteleyn S. *Hera-S: web design using sesame*. ACM Press, 2006. [cited at p. 15, 16, 18, 42, 46, 66]

Appendices

Appendix A

Application Metamodel

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xml:base="http://wwwis.win.tue.nl/~hera/Hera-S/am-metamodel.owl"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<!-- Classes -->
  <owl:Class rdf:about="#AmBasicElement"/>
  <owl:Class rdf:about="#Attribute">
    <rdfs:subClassOf rdf:resource="#AmBasicElement"/>
  </owl:Class>
  <owl:Class rdf:about="#Button">
    <rdfs:subClassOf rdf:resource="#FormElement"/>
  </owl:Class>
  <owl:Class rdf:about="#ChoiceInput">
    <rdfs:subClassOf rdf:resource="#FormElement"/>
  </owl:Class>
  <owl:Class rdf:about="#ConditionalQuery">
    <rdfs:subClassOf rdf:resource="#Query"/>
  </owl:Class>
  <owl:Class rdf:about="#ConditionalRelationship">
    <rdfs:subClassOf rdf:resource="#Relationship"/>
  </owl:Class>
  <owl:Class rdf:about="#ExtendedElement"/>
  <owl:Class rdf:about="#FormElement">
    <rdfs:subClassOf rdf:resource="#ExtendedElement"/>
  </owl:Class>
  <owl:Class rdf:about="#FormUnit">
    <rdfs:subClassOf rdf:resource="#NavigationalUnit"/>
  </owl:Class>
  <owl:Class rdf:about="#LogicUnit">
    <rdfs:subClassOf rdf:resource="#NavigationalUnit"/>
  </owl:Class>
  <owl:Class rdf:about="#NavigationalUnit">
    <rdfs:subClassOf rdf:resource="#AmBasicElement"/>
  </owl:Class>
  <owl:Class rdf:about="#Query"/>
  <owl:Class rdf:about="#Relationship">
    <rdfs:subClassOf rdf:resource="#AmBasicElement"/>
  </owl:Class>
  <owl:Class rdf:about="#Script">
    <rdfs:subClassOf rdf:resource="#ExtendedElement"/>
  </owl:Class>
  <owl:Class rdf:about="#SetUnit">
    <rdfs:subClassOf rdf:resource="#SubUnit"/>
```

```
  </owl:Class>
  <owl:Class rdf:about="#SubUnit">
    <rdfs:subClassOf rdf:resource="#UnitElement"/>
  </owl:Class>
  <owl:Class rdf:about="#TextInput">
    <rdfs:subClassOf rdf:resource="#FormElement"/>
  </owl:Class>
  <owl:Class rdf:about="#TourUnit">
    <rdfs:subClassOf rdf:resource="#SubUnit"/>
  </owl:Class>
  <owl:Class rdf:about="#UnitElement">
    <rdfs:subClassOf rdf:resource="#ExtendedElement"/>
  </owl:Class>
  <owl:Class rdf:about="#UpdateQuery">
    <rdfs:subClassOf rdf:resource="#Query"/>
  </owl:Class>
  <owl:Class rdf:about="#Variable">
    <rdfs:subClassOf rdf:resource="#UnitElement"/>
  </owl:Class>
  <owl:Class rdf:about="#WebService">
    <rdfs:subClassOf rdf:resource="#ExtendedElement"/>
  </owl:Class>

<!-- Datatypes -->
  <rdfs:Datatype rdf:about="&xsd;boolean"/>
  <rdfs:Datatype rdf:about="&xsd;string"/>

<!-- Datatype Properties -->
  <owl:DatatypeProperty rdf:about="#buttonText">
    <rdfs:domain rdf:resource="#Button"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#else">
    <rdfs:domain rdf:resource="#ConditionalQuery"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#hasAssignment">
    <rdfs:domain rdf:resource="#Relationship"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#hasCondition">
    <rdfs:domain rdf:resource="#ConditionalRelationship"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#hasDefaultCondition">
    <rdfs:domain rdf:resource="#ConditionalRelationship"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#hasDefaultValue">
    <rdfs:domain rdf:resource="#Variable"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#hasMediaType">
    <rdfs:domain rdf:resource="#Attribute"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#hasQuery">
    <rdfs:domain rdf:resource="#Attribute"/>
    <rdfs:domain rdf:resource="#Relationship"/>
    <rdfs:domain rdf:resource="#SubUnit"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#hasTargetFormat">
    <rdfs:domain rdf:resource="#Script"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#if">
    <rdfs:domain rdf:resource="#ConditionalQuery"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
```

```
  <owl:DatatypeProperty rdf:about="#option">
    <rdfs:domain rdf:resource="#ChoiceInput"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#queryBody">
    <rdfs:domain rdf:resource="#Query"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#queryType">
    <rdfs:domain rdf:resource="#Query"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#then">
    <rdfs:domain rdf:resource="#ConditionalQuery"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#varName">
    <rdfs:domain rdf:resource="#Variable"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>

<!-- Object Properties -->
  <owl:ObjectProperty rdf:about="#hasAttribute">
    <rdfs:domain rdf:resource="#NavigationalUnit"/>
    <rdfs:range rdf:resource="#Attribute"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasBinding">
    <rdfs:domain rdf:resource="#FormElement"/>
    <rdfs:range rdf:resource="#Variable"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasConditionalQuery">
    <rdfs:domain rdf:resource="#Attribute"/>
    <rdfs:domain rdf:resource="#Relationship"/>
    <rdfs:domain rdf:resource="#SubUnit"/>
    <rdfs:range rdf:resource="#ConditionalQuery"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasFormElement">
    <rdfs:domain rdf:resource="#FormUnit"/>
    <rdfs:range rdf:resource="#FormElement"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasFormType">
    <rdfs:domain rdf:resource="#FormElement"/>
    <rdfs:range rdf:resource="#Button"/>
    <rdfs:range rdf:resource="#ChoiceInput"/>
    <rdfs:range rdf:resource="#TextInput"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasFormUnit">
    <rdfs:domain rdf:resource="#NavigationalUnit"/>
    <rdfs:range rdf:resource="#FormUnit"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasInput">
    <rdfs:domain rdf:resource="#NavigationalUnit"/>
    <rdfs:range rdf:resource="#Variable"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasQueryObject">
    <rdfs:domain rdf:resource="#Attribute"/>
    <rdfs:domain rdf:resource="#Relationship"/>
    <rdfs:domain rdf:resource="#SubUnit"/>
    <rdfs:range rdf:resource="#Query"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasRelationship">
    <rdfs:domain rdf:resource="#NavigationalUnit"/>
    <rdfs:range rdf:resource="#Relationship"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasScript">
    <rdfs:domain rdf:resource="#NavigationalUnit"/>
    <rdfs:range rdf:resource="#Script"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasSetUnit">
    <rdfs:range rdf:resource="#SetUnit"/>
```

```
    <rdfs:subPropertyOf rdf:resource="#hasUnit"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasTourUnit">
    <rdfs:range rdf:resource="#TourUnit"/>
    <rdfs:subPropertyOf rdf:resource="#hasUnit"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasUnit">
    <rdfs:domain rdf:resource="#NavigationalUnit"/>
    <rdfs:range rdf:resource="#SubUnit"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasWebService">
    <rdfs:domain rdf:resource="#NavigationalUnit"/>
    <rdfs:range rdf:resource="#WebService"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#onLoadQuery">
    <rdfs:domain rdf:resource="#NavigationalUnit"/>
    <rdfs:range rdf:resource="#UpdateQuery"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#refersTo">
    <rdfs:domain rdf:resource="#Relationship"/>
    <rdfs:domain rdf:resource="#SubUnit"/>
    <rdfs:range rdf:resource="#NavigationalUnit"/>
  </owl:ObjectProperty>
  <owl:FunctionalProperty rdf:about="#sourceUnit">
    <rdf:type rdf:resource="&owl;ObjectProperty"/>
    <rdfs:domain rdf:resource="#Relationship"/>
    <rdfs:range rdf:resource="#SubUnit"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:about="#varType">
    <rdf:type rdf:resource="&owl;ObjectProperty"/>
    <rdfs:domain rdf:resource="#Variable"/>
  </owl:FunctionalProperty>
</rdf:RDF>}
```

Appendix B

IMDB Example

B.1 Imdb server configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<application name="imdb">
    <result_path>imdb_amp.rdf</result_path>
    <amp_context>"http://wwwis.win.tue.nl/~hera/Hera-S/amp#"</amp_context>
    <app_server>http://localhost:8080/herasWeb/HydragenWeb</app_server>
    <pmpath>nl\tue\heras\examples\imdb1\imdb_pm.xsl</pmpath>
    <amspath>nl\tue\heras\metamodel\am-metamodel.owl</amspath>
    <ampath>nl\tue\heras\examples\imdb1\imdb_am.owl</ampath>
    <servers>
        <dm_server_info>
           <server_path>
              http://localhost:8080/openrdf-http-server-2.0-beta5
           </server_path>
           <repository_id>imdb</repository_id>
        </dm_server_info>
        <cm_server_info>
           <server_path>
              http://localhost:8080/openrdf-http-server-2.0-beta5
           </server_path>
           <repository_id>imdb</repository_id>
        </cm_server_info>
        <content_server_info>
           <server_path>
              http://localhost:8080/openrdf-http-server-2.0-beta5
           </server_path>
            <repository_id>imdb</repository_id>
        </content_server_info>
    </servers>
</application>
```

B.2 Imdb application model

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY am-metamodel
                   "http://wwwis.win.tue.nl/~hera/Hera-S/am-metamodel.owl#">
  <!ENTITY imdb_am "http://wwwis.win.tue.nl/~hera/Hera-S/imdb_am.owl#">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
```

```
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&imdb_am;"
         xmlns:am-metamodel="&am-metamodel;"
         xmlns:owl="&owl;"
         xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;">

<!-- LoginUnit -->
    <am-metamodel:NavigationalUnit rdf:about="&imdb_am;LoginUnit">
     <am-metamodel:hasInput rdf:resource="&imdb_am;LoginFailure"/>
     <am-metamodel:hasAttribute rdf:resource="&imdb_am;LoginFailureMessage"/>
      <am-metamodel:hasAttribute rdf:resource="&imdb_am;LoginLabel"/>
      <am-metamodel:hasFormUnit rdf:resource="&imdb_am;LoginForm"/>
      <am-metamodel:hasRelationship rdf:resource="&imdb_am;RegisterRelationship"/>
    </am-metamodel:NavigationalUnit>

    <am-metamodel:FormUnit rdf:about="&imdb_am;LoginForm">
     <am-metamodel:hasFormElement rdf:resource="&imdb_am;LoginButton"/>
     <am-metamodel:hasFormElement rdf:resource="&imdb_am;LoginUserNameInput"/>
     <am-metamodel:hasFormElement rdf:resource="&imdb_am;LoginPasswordInput"/>
      <am-metamodel:hasRelationship rdf:resource="&imdb_am;LoginRelationship"/>
    </am-metamodel:FormUnit>

    <am-metamodel:Attribute rdf:about="&imdb_am;LoginLabel">
      <rdfs:label rdf:datatype="&xsd;string">Log in</rdfs:label>
    </am-metamodel:Attribute>

  <am-metamodel:Attribute rdf:about="&imdb_am;LoginFailureMessage">
    <rdfs:label rdf:datatype="&xsd;string">LoginFailureMessage</rdfs:label>
    <am-metamodel:hasConditionalQuery rdf:resource="&imdb_am;LoginFailureConditionalQuery"/>
  </am-metamodel:Attribute>

  <am-metamodel:ConditionalQuery
       rdf:about="&imdb_am;LoginFailureConditionalQuery">
    <am-metamodel:else rdf:datatype="&xsd;string"></am-metamodel:else>
    <am-metamodel:if rdf:datatype="&xsd;string">"$LoginFailure$"=="true"
    </am-metamodel:if>
    <am-metamodel:then rdf:datatype="&xsd;string">"Login failed, please retry!"
    </am-metamodel:then>
  </am-metamodel:ConditionalQuery>

  <am-metamodel:Button rdf:about="&imdb_am;LoginButton">
    <am-metamodel:buttonText rdf:datatype="&xsd;string">Login!
    </am-metamodel:buttonText>
  </am-metamodel:Button>

  <am-metamodel:TextInput rdf:about="&imdb_am;LoginUserNameInput">
    <am-metamodel:hasBinding rdf:resource="&imdb_am;loginName"/>
  </am-metamodel:TextInput>

   <am-metamodel:TextInput rdf:about="&imdb_am;LoginPasswordInput">
    <am-metamodel:hasBinding rdf:resource="&imdb_am;loginPassword"/>
  </am-metamodel:TextInput>

  <am-metamodel:Relationship rdf:about="&imdb_am;LoginRelationship">
    <am-metamodel:refersTo rdf:resource="&imdb_am;LoginPassOrFailUnit"/>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT U FROM {U} rdf:type {imdb_cm:RegisteredUser};
                         imdb_cm:hasLoginName {name};
                         imdb_cm:hasLoginPassword {passwd}
       WHERE (name LIKE "$loginName$") and (passwd LIKE "$loginPassword$")
       USING NAMESPACE
       imdb_cm = &lt;http://wwwis.win.tue.nl/~hera/Hera-S/imdb_cm.owl#&gt;,
       imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#&gt;,
       imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#&gt;
    </am-metamodel:hasQuery>
  </am-metamodel:Relationship>

  <am-metamodel:Variable rdf:about="&imdb_am;loginName">
```

```
    <am-metamodel:varName rdf:datatype="&xsd;string">loginName
    </am-metamodel:varName>
  </am-metamodel:Variable>

  <am-metamodel:Variable rdf:about="&imdb_am;loginPassword">
    <am-metamodel:varName rdf:datatype="&xsd;string">loginPassword
    </am-metamodel:varName>
  </am-metamodel:Variable>

  <am-metamodel:Relationship rdf:about="&imdb_am;RegisterRelationship">
    <am-metamodel:refersTo rdf:resource="&imdb_am;RegistrationUnit"/>
  </am-metamodel:Relationship>

<am-metamodel:Variable rdf:about="&imdb_am;LoginFailure">
    <am-metamodel:varName rdf:datatype="&xsd;string">LoginFailure
    </am-metamodel:varName>
    <am-metamodel:hasDefaultValue rdf:datatype="&xsd;string">false
    </am-metamodel:hasDefaultValue>
  </am-metamodel:Variable>
<!-- end LoginUnit -->

<!-- LoginPassOrFailUnit -->
    <am-metamodel:LogicUnit rdf:about="&imdb_am;LoginPassOrFailUnit">
     <am-metamodel:hasRelationship rdf:resource="&imdb_am;LoginPassRelationship"/>
      <am-metamodel:hasRelationship rdf:resource="&imdb_am;LoginFailureRelationship"/>
    </am-metamodel:LogicUnit>

   <am-metamodel:ConditionalRelationship rdf:about="&imdb_am;LoginPassRelationship">
   <am-metamodel:refersTo rdf:resource="&imdb_am;SearchUnit"/>
   <am-metamodel:hasCondition rdf:datatype="&xsd;string">"$U$"!=""
      </am-metamodel:hasCondition>
   </am-metamodel:ConditionalRelationship>

   <am-metamodel:ConditionalRelationship rdf:about="&imdb_am;LoginFailureRelationship">
      <am-metamodel:refersTo rdf:resource="&imdb_am;LoginUnit"/>
      <am-metamodel:hasDefaultCondition rdf:datatype="&xsd;string">default
      </am-metamodel:hasDefaultCondition>
   <am-metamodel:hasAssignment rdf:datatype="&xsd;string">LoginFailure=true
      </am-metamodel:hasAssignment>
      </am-metamodel:ConditionalRelationship>

<!-- RegistrationUnit -->
    <am-metamodel:NavigationalUnit rdf:about="&imdb_am;RegistrationUnit">
     <am-metamodel:hasAttribute rdf:resource="&imdb_am;RegisterLabel"/>
      <am-metamodel:hasFormUnit rdf:resource="&imdb_am;RegisterForm"/>
    </am-metamodel:NavigationalUnit>

  <am-metamodel:Attribute rdf:about="&imdb_am;RegisterLabel">
    <rdfs:label rdf:datatype="&xsd;string">
      Fill out all the fields below. Hit "Register!"
    </rdfs:label>
  </am-metamodel:Attribute>

   <am-metamodel:FormUnit rdf:about="&imdb_am;RegisterForm">
     <am-metamodel:hasFormElement rdf:resource="&imdb_am;RegisterButton"/>
     <am-metamodel:hasFormElement rdf:resource="&imdb_am;RegisterGenderChoice"/>
     <am-metamodel:hasFormElement rdf:resource="&imdb_am;RegisterUserNameInput"/>
     <am-metamodel:hasFormElement rdf:resource="&imdb_am;RegisterLoginNameInput"/>
     <am-metamodel:hasFormElement rdf:resource="&imdb_am;RegisterPasswordInput"/>
     <am-metamodel:hasFormElement rdf:resource="&imdb_am;RegisterAgeInput"/>
    <am-metamodel:hasRelationship rdf:resource="&imdb_am;RegisterFormRelationship"/>
   </am-metamodel:FormUnit>

  <am-metamodel:TextInput rdf:about="&imdb_am;RegisterUserNameInput">
    <am-metamodel:hasBinding rdf:resource="&imdb_am;usernameTerm"/>
  </am-metamodel:TextInput>

  <am-metamodel:TextInput rdf:about="&imdb_am;RegisterLoginNameInput">
    <am-metamodel:hasBinding rdf:resource="&imdb_am;loginNameTerm"/>
  </am-metamodel:TextInput>
```

```
  <am-metamodel:TextInput rdf:about="&imdb_am;RegisterPasswordInput">
    <am-metamodel:hasBinding rdf:resource="&imdb_am;passwordTerm"/>
  </am-metamodel:TextInput>

  <am-metamodel:TextInput rdf:about="&imdb_am;RegisterAgeInput">
    <am-metamodel:hasBinding rdf:resource="&imdb_am;ageTerm"/>
  </am-metamodel:TextInput>

  <am-metamodel:ChoiceInput rdf:about="&imdb_am;RegisterGenderChoice">
    <am-metamodel:option  rdf:datatype="&xsd;string">Male</am-metamodel:option>
    <am-metamodel:option  rdf:datatype="&xsd;string">Female
    </am-metamodel:option>
    <am-metamodel:hasBinding rdf:resource="&imdb_am;userGender"/>
  </am-metamodel:ChoiceInput>

  <am-metamodel:Button rdf:about="&imdb_am;RegisterButton">
    <am-metamodel:buttonText rdf:datatype="&xsd;string">Register
    </am-metamodel:buttonText>
  </am-metamodel:Button>

  <am-metamodel:Relationship rdf:about="&imdb_am;RegisterFormRelationship">
    <am-metamodel:refersTo rdf:resource="&imdb_am;RegisterLogicUnit"/>
  </am-metamodel:Relationship>

  <am-metamodel:Variable rdf:about="&imdb_am;usernameTerm">
    <am-metamodel:varName rdf:datatype="&xsd;string">userName
    </am-metamodel:varName>
  </am-metamodel:Variable>

  <am-metamodel:Variable rdf:about="&imdb_am;loginNameTerm">
    <am-metamodel:varName rdf:datatype="&xsd;string">loginName
    </am-metamodel:varName>
  </am-metamodel:Variable>

  <am-metamodel:Variable rdf:about="&imdb_am;passwordTerm">
    <am-metamodel:varName rdf:datatype="&xsd;string">loginPassword
    </am-metamodel:varName>
  </am-metamodel:Variable>

  <am-metamodel:Variable rdf:about="&imdb_am;ageTerm">
    <am-metamodel:varName rdf:datatype="&xsd;string">userAge
    </am-metamodel:varName>
  </am-metamodel:Variable>

  <am-metamodel:Variable rdf:about="&imdb_am;userGender">
    <am-metamodel:varName rdf:datatype="&xsd;string">userGender
    </am-metamodel:varName>
  </am-metamodel:Variable>
<!-- end RegistrationUnit -->

<!-- RegisterLogicUnit -->
  <am-metamodel:LogicUnit rdf:about="&imdb_am;RegisterLogicUnit">
   <am-metamodel:onLoadQuery rdf:resource="&imdb_am;newUserQuery"/>
    <am-metamodel:hasRelationship rdf:resource="&imdb_am;RegisterLoginRelationship"/>
  </am-metamodel:LogicUnit>

  <am-metamodel:UpdateQuery rdf:about="&imdb_am;newUserQuery">
    <am-metamodel:queryBody rdf:datatype="&xsd;string">
       Construct {} rdf:type {imdb_cm:RegisteredUser};
                    imdb_cm:hasName {"$userName$"};
                    imdb_cm:hasLoginName {"$loginName$"};
                    imdb_cm:hasLoginPassword {"$loginPassword$"};
                    imdb_cm:age {$userAge$};
                    imdb_cm:gender {"$userGender$"}
      USING NAMESPACE
            imdb_cm = &lt;http://wwwis.win.tue.nl/~hera/Hera-S/imdb_cm.owl#>,
            imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
            imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:queryBody>
```

```
    <am-metamodel:queryType rdf:datatype="&xsd;string">SERQL
    </am-metamodel:queryType>
  </am-metamodel:UpdateQuery>

  <am-metamodel:Relationship rdf:about="&imdb_am;RegisterLoginRelationship">
    <am-metamodel:refersTo rdf:resource="&imdb_am;LoginUnit"/>
  </am-metamodel:Relationship>

<!-- SearchUnit -->
    <am-metamodel:NavigationalUnit rdf:about="&imdb_am;SearchUnit">
     <am-metamodel:hasInput rdf:resource="&imdb_am;UserVar"/>
     <am-metamodel:hasAttribute rdf:resource="&imdb_am;WelcomeLabel"/>
      <am-metamodel:hasAttribute rdf:resource="&imdb_am;SearchLabel"/>
      <am-metamodel:hasFormUnit rdf:resource="&imdb_am;SearchForm"/>
    </am-metamodel:NavigationalUnit>

    <am-metamodel:FormUnit rdf:about="&imdb_am;SearchForm">
     <am-metamodel:hasAttribute rdf:resource="&imdb_am;SearchLabel"/>
     <am-metamodel:hasFormElement rdf:resource="&imdb_am;SearchButton"/>
     <am-metamodel:hasFormElement rdf:resource="&imdb_am;SearchInput"/>
     <am-metamodel:hasFormElement rdf:resource="&imdb_am;SearchChoice"/>
    <am-metamodel:hasRelationship rdf:resource="&imdb_am;SearchRelationship"/>
   </am-metamodel:FormUnit>

  <am-metamodel:Attribute rdf:about="&imdb_am;SearchLabel">
    <rdfs:label rdf:datatype="&xsd;string">Search</rdfs:label>
  </am-metamodel:Attribute>

<am-metamodel:Attribute rdf:about="&imdb_am;WelcomeLabel">
    <rdfs:label rdf:datatype="&xsd;string">Welcome</rdfs:label>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
      SELECT UserName FROM {$U$} rdf:type {imdb_cm:RegisteredUser};
                                imdb_cm:hasName {UserName}
      using namespace
         imdb_cm = &lt;http://wwwis.win.tue.nl/~hera/Hera-S/imdb_cm.owl#>,
         imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
         imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
  </am-metamodel:Attribute>

  <am-metamodel:Button rdf:about="&imdb_am;SearchButton">
    <am-metamodel:buttonText rdf:datatype="&xsd;string">Go
    </am-metamodel:buttonText>
  </am-metamodel:Button>

  <am-metamodel:TextInput rdf:about="&imdb_am;SearchInput">
    <am-metamodel:hasBinding rdf:resource="&imdb_am;searchTerm"/>
  </am-metamodel:TextInput>

  <am-metamodel:ChoiceInput rdf:about="&imdb_am;SearchChoice">
    <am-metamodel:option rdf:datatype="&xsd;string">Movie</am-metamodel:option>
    <am-metamodel:option rdf:datatype="&xsd;string">Actor</am-metamodel:option>
    <am-metamodel:hasBinding rdf:resource="&imdb_am;searchCategory"/>
  </am-metamodel:ChoiceInput>

  <am-metamodel:Relationship rdf:about="&imdb_am;SearchRelationship">
    <am-metamodel:refersTo rdf:resource="&imdb_am;SearchResultUnit"/>
  </am-metamodel:Relationship>

  <am-metamodel:Variable rdf:about="&imdb_am;searchTerm">
    <am-metamodel:varName rdf:datatype="&xsd;string">searchTerm
    </am-metamodel:varName>
  </am-metamodel:Variable>

  <am-metamodel:Variable rdf:about="&imdb_am;searchCategory">
    <am-metamodel:varName rdf:datatype="&xsd;string">searchCategory
    </am-metamodel:varName>
  </am-metamodel:Variable>

<am-metamodel:Variable rdf:about="&imdb_am;UserVar">
```

```
    <am-metamodel:varName rdf:datatype="&xsd;string">U</am-metamodel:varName>
  </am-metamodel:Variable>

<!-- SearchResultUnit -->
  <am-metamodel:LogicUnit rdf:about="&imdb_am;SearchResultUnit">
    <am-metamodel:hasRelationship rdf:resource="&imdb_am;MovieSearchConditionalRelationship"/>
    <am-metamodel:hasRelationship rdf:resource="&imdb_am;ActorSearchConditionalRelationship"/>
  </am-metamodel:LogicUnit>

  <am-metamodel:ConditionalRelationship rdf:about="&imdb_am;MovieSearchConditionalRelationship">
    <am-metamodel:hasCondition rdf:datatype="&xsd;string">"$searchCategory$"=="Movie"
    </am-metamodel:hasCondition>
 <am-metamodel:refersTo rdf:resource="&imdb_am;MovieSearchResultUnit"/>
  </am-metamodel:ConditionalRelationship>

  <am-metamodel:ConditionalRelationship
       rdf:about="&imdb_am;ActorSearchConditionalRelationship">
    <am-metamodel:hasCondition rdf:datatype="&xsd;string">"$searchCategory$"=="Actor"
    </am-metamodel:hasCondition>
 <am-metamodel:refersTo rdf:resource="&imdb_am;ActorSearchResultUnit"/>
  </am-metamodel:ConditionalRelationship>

<!-- MovieSearchResultUnit -->
<am-metamodel:NavigationalUnit rdf:about="&imdb_am;MovieSearchResultUnit">
    <am-metamodel:hasSetUnit rdf:resource="&imdb_am;MovieSearchResultSet"/>
  </am-metamodel:NavigationalUnit>

  <am-metamodel:SetUnit rdf:about="&imdb_am;MovieSearchResultSet">
    <rdfs:comment rdf:datatype="&xsd;string">Results</rdfs:comment>
    <am-metamodel:refersTo rdf:resource="&imdb_am;MovieSearchResultElementUnit"/>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
        SELECT M FROM {M} rdf:type {imdb_dm:Movie}; rdfs:label {X}
        WHERE X Like "*$searchTerm$*" ignore case
        using namespace
          imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#&gt;
    </am-metamodel:hasQuery>
  </am-metamodel:SetUnit>

  <am-metamodel:NavigationalUnit rdf:about="&imdb_am;MovieSearchResultElementUnit">
    <am-metamodel:hasInput rdf:resource="&imdb_am;MovieResultInputVar"/>
    <am-metamodel:hasAttribute  rdf:resource="&imdb_am;MovieUnitTitle"/>
    <am-metamodel:hasAttribute  rdf:resource="&imdb_am;MovieUnitYear"/>
    <am-metamodel:hasRelationship rdf:resource="&imdb_am;MovieUnitResultRelationship"/>
  </am-metamodel:NavigationalUnit>

  <am-metamodel:Relationship rdf:about="&imdb_am;MovieUnitResultRelationship">
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT M FROM {M} rdf:type {imdb_dm:Movie}
       where M = $M$
       using namespace
       imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
       imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
    <am-metamodel:refersTo rdf:resource="&imdb_am;MovieMainUnit"/>
  </am-metamodel:Relationship>

  <am-metamodel:Variable rdf:about="&imdb_am;MovieResultInputVar">
    <am-metamodel:varName rdf:datatype="&xsd;string">M</am-metamodel:varName>
  </am-metamodel:Variable>
<!-- end  MovieSearchResultUnit -->

<!-- ActorSearchResultUnit -->
<am-metamodel:NavigationalUnit rdf:about="&imdb_am;ActorSearchResultUnit">
    <am-metamodel:hasSetUnit rdf:resource="&imdb_am;ActorSearchResultSet"/>
  </am-metamodel:NavigationalUnit>

<am-metamodel:SetUnit rdf:about="&imdb_am;ActorSearchResultSet">
    <rdfs:comment rdf:datatype="&xsd;string">Results</rdfs:comment>
    <am-metamodel:refersTo rdf:resource="&imdb_am;ActorSearchResultElementUnit"/>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
```

```
      SELECT A FROM {A} rdf:type {imdb_dm:Actor};
                        rdfs:label {X}
      where X Like "*$searchTerm$*"
      using namespace
        imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
        imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#&gt;
    </am-metamodel:hasQuery>
  </am-metamodel:SetUnit>

  <am-metamodel:NavigationalUnit rdf:about="&imdb_am;ActorSearchResultElementUnit">
    <am-metamodel:hasInput rdf:resource="&imdb_am;ActorResultInputVar"/>
    <am-metamodel:hasAttribute  rdf:resource="&imdb_am;ActorUnitLabel"/>
    <am-metamodel:hasAttribute  rdf:resource="&imdb_am;ActorUnitDob"/>
    <am-metamodel:hasRelationship rdf:resource="&imdb_am;ActorUnitResultRelationship"/>
  </am-metamodel:NavigationalUnit>

  <am-metamodel:Relationship rdf:about="&imdb_am;ActorUnitResultRelationship">
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT A FROM {A} rdf:type {imdb_dm:Actor}
       where A = $A$
       using namespace
         imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
         imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
    <am-metamodel:refersTo rdf:resource="&imdb_am;ActorMainUnit"/>
  </am-metamodel:Relationship>

   <am-metamodel:Variable rdf:about="&imdb_am;ActorResultInputVar">
    <am-metamodel:varName rdf:datatype="&xsd;string">A</am-metamodel:varName>
  </am-metamodel:Variable>

<!-- end ActorSearchResultUnit -->

<!-- ActorMainUnit -->
  <am-metamodel:NavigationalUnit rdf:about="&imdb_am;ActorMainUnit">
    <am-metamodel:hasUnit rdf:resource="&imdb_am;ActorReferredSubUnit"/>
    <am-metamodel:hasUnit rdf:resource="&imdb_am;ActorMenuSubUnit"/>
  </am-metamodel:NavigationalUnit>
<!-- end ActorMainUnit -->

<!-- ActorReferredSubUnit -->
  <am-metamodel:SubUnit rdf:about="&imdb_am;ActorReferredSubUnit">
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT A FROM {A} rdf:type {imdb_dm:Actor}
       where A = $A$
       using namespace
         imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
         imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
    <am-metamodel:refersTo rdf:resource="&imdb_am;ActorUnit"/>
  </am-metamodel:SubUnit>
<!-- ActorReferredUnit -->

<!-- ActorMenuSubUnit -->
  <am-metamodel:SubUnit rdf:about="&imdb_am;ActorMenuSubUnit">
    <am-metamodel:refersTo rdf:resource="&imdb_am;ActorMenuNavigationUnit"/>
  </am-metamodel:SubUnit>
<!-- ActorMenuSubUnit -->

<!-- ActorMenuNavigationUnit -->
  <am-metamodel:NavigationalUnit rdf:about="&imdb_am;ActorMenuNavigationUnit">
    <am-metamodel:hasUnit rdf:resource="&imdb_am;SearchRefUnit"/>
    <am-metamodel:hasSetUnit rdf:resource="&imdb_am;ActorUnitUserFavoritesSetUnit"/>
  </am-metamodel:NavigationalUnit>

  <am-metamodel:SetUnit rdf:about="&imdb_am;ActorUnitUserFavoritesSetUnit">
    <rdfs:comment rdf:datatype="&xsd;string">Favorites</rdfs:comment>
    <am-metamodel:refersTo rdf:resource="&imdb_am;ActorUnitUserFavoritesUnit"/>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT DISTINCT ActorFavElement FROM {$U$} imdb_cm:hasFavoriteActor {ActorFavElement}
```

```
      using namespace
        imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
        imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>,
        imdb_cm = &lt;http://wwwis.win.tue.nl/~hera/Hera-S/imdb_cm.owl#>
    </am-metamodel:hasQuery>
  </am-metamodel:SetUnit>

  <am-metamodel:NavigationalUnit rdf:about="&imdb_am;ActorUnitUserFavoritesUnit">
    <am-metamodel:hasAttribute  rdf:resource="&imdb_am;FavActorName"/>
    <am-metamodel:hasRelationship rdf:resource="&imdb_am;ActorFavSetRelationship"/>
  </am-metamodel:NavigationalUnit>

  <am-metamodel:Attribute rdf:about="&imdb_am;FavActorName">
    <rdfs:label rdf:datatype="&xsd;string">Actor Name</rdfs:label>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT T FROM {$ActorFavElement$} rdf:type {imdb_dm:Actor};
                                         rdfs:label {T}
       using namespace
         imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
         imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
  </am-metamodel:Attribute>

  <am-metamodel:Relationship rdf:about="&imdb_am;ActorFavSetRelationship">
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT A FROM {A} rdf:type {imdb_dm:Actor}
       where A = $ActorFavElement$
       using namespace
         imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
         imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
    <am-metamodel:refersTo rdf:resource="&imdb_am;ActorMainUnit"/>
    <am-metamodel:sourceUnit rdf:resource="&imdb_am;ActorReferredSubUnit"/>
  </am-metamodel:Relationship>
<!-- ActorMenuNavigationUnit -->


<!-- Actor Unit -->
  <am-metamodel:NavigationalUnit rdf:about="&imdb_am;ActorUnit">
    <am-metamodel:hasAttribute rdf:resource="&imdb_am;ActorUnitLabel"/>
    <am-metamodel:hasAttribute rdf:resource="&imdb_am;ActorUnitDob"/>
    <am-metamodel:hasAttribute rdf:resource="&imdb_am;ActorUnitPob"/>
    <am-metamodel:hasAttribute rdf:resource="&imdb_am;ActorUnitCob"/>
    <am-metamodel:hasAttribute rdf:resource="&imdb_am;ActorUnitminiBio"/>
    <am-metamodel:hasAttribute rdf:resource="&imdb_am;ActorUnitPhoto"/>
    <am-metamodel:hasInput rdf:resource="&imdb_am;ActorUnitVar"/>
    <am-metamodel:hasSetUnit rdf:resource="&imdb_am;ActorUnitMoviesPlayedIn"/>
    <am-metamodel:hasRelationship rdf:resource="&imdb_am;ActorUnitAddToFavorites"/>
  </am-metamodel:NavigationalUnit>

  <am-metamodel:Attribute rdf:about="&imdb_am;ActorUnitLabel">
    <rdfs:label rdf:datatype="&xsd;string">Name</rdfs:label>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT N FROM {$A$} rdf:type {imdb_dm:Actor};
                           rdfs:label {N}
       using namespace
          imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
          imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
  </am-metamodel:Attribute>

  <am-metamodel:Attribute rdf:about="&imdb_am;ActorUnitDob">
    <rdfs:label rdf:datatype="&xsd;string">Date of Birth</rdfs:label>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT D FROM {$A$} rdf:type {imdb_dm:Actor};
                           imdb_dm:birthDate {D}
       using namespace
          imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
          imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
```

```
</am-metamodel:Attribute>

<am-metamodel:Attribute rdf:about="&imdb_am;ActorUnitPob">
  <rdfs:label rdf:datatype="&xsd;string">Place of Birth</rdfs:label>
  <am-metamodel:hasQuery rdf:datatype="&xsd;string">
     SELECT L FROM {$A$} rdf:type {imdb_dm:Actor};
                         imdb_dm:birthLocation {} imdb_pl:inPlace {} rdfs:label {L}
     using namespace
        imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
        imdb_pl = &lt;http://www.vub.ac.be/place-schema#>,
        imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
  </am-metamodel:hasQuery>
</am-metamodel:Attribute>

<am-metamodel:Attribute rdf:about="&imdb_am;ActorUnitCob">
  <rdfs:label rdf:datatype="&xsd;string">Country of Birth</rdfs:label>
  <am-metamodel:hasQuery rdf:datatype="&xsd;string">
      SELECT L
      FROM {$A$} rdf:type {imdb_dm:Actor};
                 imdb_dm:birthLocation {} imdb_pl:inCountry {} rdfs:label {L}
      using namespace
          imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
          imdb_pl = &lt;http://www.vub.ac.be/place-schema#>,
          imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
  </am-metamodel:hasQuery>
</am-metamodel:Attribute>

<am-metamodel:Attribute rdf:about="&imdb_am;ActorUnitminiBio">
  <rdfs:label rdf:datatype="&xsd;string">Mini Bio</rdfs:label>
  <am-metamodel:hasQuery rdf:datatype="&xsd;string">
      SELECT B FROM {$A$} rdf:type {imdb_dm:Actor};
                          imdb_dm:miniBio {B}
      using namespace
         imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
         imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
  </am-metamodel:hasQuery>
</am-metamodel:Attribute>

<am-metamodel:Attribute rdf:about="&imdb_am;ActorUnitPhoto">
  <rdfs:label rdf:datatype="&xsd;string">Photo</rdfs:label>
  <am-metamodel:hasMediaType rdf:datatype="&xsd;string">
    image/jpg
  </am-metamodel:hasMediaType>
  <am-metamodel:hasQuery rdf:datatype="&xsd;string">
     SELECT P FROM {$A$} imdb_dm:hasMainPhoto {} imdb_dm:photoURL {P}
     using namespace
     imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
     imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
  </am-metamodel:hasQuery>
</am-metamodel:Attribute>

<am-metamodel:Variable rdf:about="&imdb_am;ActorUnitVar">
  <am-metamodel:varName rdf:datatype="&xsd;string">A</am-metamodel:varName>
</am-metamodel:Variable>

<am-metamodel:SetUnit rdf:about="&imdb_am;ActorUnitMoviesPlayedIn">
  <am-metamodel:hasConditionalQuery rdf:resource="&imdb_am;ActorUnitConditionalQuery"/>
  <am-metamodel:refersTo rdf:resource="&imdb_am;ActorUnitMoviesPlayedInSetElementUnit"/>
</am-metamodel:SetUnit>

<am-metamodel:ConditionalQuery rdf:about="&imdb_am;ActorUnitConditionalQuery">
  <am-metamodel:else rdf:datatype="&xsd;string">
     SELECT DISTINCT M
     FROM {$A$} rdf:type {imdb_dm:Actor};
                imdb_dm:actorFilmography {} imdb_dm:castMovie {M}
                imdb_dm:hasAgeRating {} rdfs:label {arate}
    WHERE arate != "NC-17"
    USING NAMESPACE
        imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
        imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>,
```

```
          imdb_cm = &lt;http://wwwis.win.tue.nl/~hera/Hera-S/imdb_cm.owl#>
    </am-metamodel:else>
    <am-metamodel:if rdf:datatype="&xsd;string">
       SELECT * FROM {$U$} imdb_cm:age {G}
       WHERE G > 17
       using namespace
       imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>,
       imdb_cm = &lt;http://wwwis.win.tue.nl/~hera/Hera-S/imdb_cm.owl#>
    </am-metamodel:if>
    <am-metamodel:then rdf:datatype="&xsd;string">
       SELECT DISTINCT M
       FROM {$A$} rdf:type {imdb_dm:Actor};
                  imdb_dm:actorFilmography {} imdb_dm:castMovie {M}
       using namespace
          imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
          imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:then>
  </am-metamodel:ConditionalQuery>

  <am-metamodel:NavigationalUnit
      rdf:about="&imdb_am;ActorUnitMoviesPlayedInSetElementUnit">
    <am-metamodel:hasAttribute rdf:resource="&imdb_am;MovieUnitTitle"/>
    <am-metamodel:hasAttribute rdf:resource="&imdb_am;MovieUnitPhoto"/>
    <am-metamodel:hasAttribute rdf:resource="&imdb_am;MovieUnitYear"/>
    <am-metamodel:hasRelationship rdf:resource="&imdb_am;MovieUnitRelationship"/>
  </am-metamodel:NavigationalUnit>

  <am-metamodel:Relationship rdf:about="&imdb_am;MovieUnitRelationship">
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT M FROM {M} rdf:type {imdb_dm:Movie}
       where M = $M$
       using namespace
         imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
         imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
    <am-metamodel:refersTo rdf:resource="&imdb_am;MovieMainUnit"/>
  </am-metamodel:Relationship>

  <am-metamodel:Relationship rdf:about="&imdb_am;ActorUnitAddToFavorites">
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT A FROM {A} rdf:type {imdb_dm:Actor}
       where A = $A$
       using namespace
         imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
         imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
    <am-metamodel:refersTo rdf:resource="&imdb_am;AddToFavoritesActorUnit"/>
  </am-metamodel:Relationship>

<!-- end ActorUnit -->


<!-- MovieMainUnit -->
  <am-metamodel:NavigationalUnit rdf:about="&imdb_am;MovieMainUnit">
    <am-metamodel:hasUnit rdf:resource="&imdb_am;MovieReferredSubUnit"/>
    <am-metamodel:hasUnit rdf:resource="&imdb_am;MovieMenuSubUnit"/>
  </am-metamodel:NavigationalUnit>
<!-- end MovieMainUnit -->

<!-- MovieReferredSubUnit -->
  <am-metamodel:SubUnit rdf:about="&imdb_am;MovieReferredSubUnit">
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
      SELECT M FROM {M} rdf:type {imdb_dm:Movie}
      where M = $M$
      using namespace
        imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
        imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
    <am-metamodel:refersTo rdf:resource="&imdb_am;MovieUnit"/>
  </am-metamodel:SubUnit>
```

```
<!-- MovieReferredUnit -->

<!-- MovieMenuSubUnit -->
  <am-metamodel:SubUnit rdf:about="&imdb_am;MovieMenuSubUnit">
    <am-metamodel:refersTo rdf:resource="&imdb_am;MovieMenuNavigationUnit"/>
  </am-metamodel:SubUnit>
<!-- MovieMenuSubUnit -->

<!-- MovieMenuNavigationUnit -->
  <am-metamodel:NavigationalUnit rdf:about="&imdb_am;MovieMenuNavigationUnit">
    <am-metamodel:hasUnit rdf:resource="&imdb_am;SearchRefUnit"/>
    <am-metamodel:hasSetUnit rdf:resource="&imdb_am;MovieUnitTenSetUnit"/>
    <am-metamodel:hasSetUnit rdf:resource="&imdb_am;MovieUnitUserFavoritesSetUnit"/>
  </am-metamodel:NavigationalUnit>

    <am-metamodel:SetUnit rdf:about="&imdb_am;MovieUnitTenSetUnit">
    <rdfs:comment rdf:datatype="&xsd;string">Top 10 Movies</rdfs:comment>
    <am-metamodel:refersTo rdf:resource="&imdb_am;MovieUnitTenSetElementUnit"/>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT DISTINCT MovieSetElement
       FROM {MovieSetElement} rdf:type {imdb_dm:Movie};
                              imdb_dm:hasUserRating {} imdb_dm:averageVote {X}
       where X > 6.0 limit 10
       using namespace
         imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
         imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>,
         imdb_cm = &lt;http://wwwis.win.tue.nl/~hera/Hera-S/imdb_cm.owl#>
    </am-metamodel:hasQuery>
  </am-metamodel:SetUnit>

  <am-metamodel:NavigationalUnit
       rdf:about="&imdb_am;MovieUnitTenSetElementUnit">
    <am-metamodel:hasAttribute  rdf:resource="&imdb_am;MovieName"/>
    <am-metamodel:hasRelationship rdf:resource="&imdb_am;MovieTenSetRelationship"/>
  </am-metamodel:NavigationalUnit>

  <am-metamodel:Attribute rdf:about="&imdb_am;MovieName">
    <rdfs:label rdf:datatype="&xsd;string">Movie Name</rdfs:label>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT T FROM {$MovieSetElement$} rdf:type {imdb_dm:Movie};
                                         rdfs:label {T}
       using namespace
         imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
         imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
  </am-metamodel:Attribute>

  <am-metamodel:Relationship rdf:about="&imdb_am;MovieTenSetRelationship">
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT M FROM {M} rdf:type {imdb_dm:Movie}
       where M = $MovieSetElement$
       using namespace
         imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
         imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
    <am-metamodel:refersTo rdf:resource="&imdb_am;MovieMainUnit"/>
    <am-metamodel:sourceUnit rdf:resource="&imdb_am;MovieReferredSubUnit"/>
  </am-metamodel:Relationship>

  <am-metamodel:SetUnit rdf:about="&imdb_am;MovieUnitUserFavoritesSetUnit">
    <rdfs:comment rdf:datatype="&xsd;string">Favorites</rdfs:comment>
    <am-metamodel:refersTo rdf:resource="&imdb_am;MovieUnitUserFavoritesUnit"/>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT DISTINCT MovieFavElement
       FROM {$U$} imdb_cm:hasFavorite {MovieFavElement}
       using namespace
         imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
         imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>,
         imdb_cm = &lt;http://wwwis.win.tue.nl/~hera/Hera-S/imdb_cm.owl#>
    </am-metamodel:hasQuery>
```

```
  </am-metamodel:SetUnit>

  <am-metamodel:NavigationalUnit  rdf:about="&imdb_am;MovieUnitUserFavoritesUnit">
    <am-metamodel:hasAttribute  rdf:resource="&imdb_am;FavMovieName"/>
    <am-metamodel:hasRelationship rdf:resource="&imdb_am;MovieFavSetRelationship"/>
  </am-metamodel:NavigationalUnit>

  <am-metamodel:Attribute rdf:about="&imdb_am;FavMovieName">
    <rdfs:label rdf:datatype="&xsd;string">Movie Name</rdfs:label>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT T FROM {$MovieFavElement$} rdf:type {imdb_dm:Movie};
                                         rdfs:label {T}
       using namespace
          imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
          imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
     </am-metamodel:hasQuery>
  </am-metamodel:Attribute>

  <am-metamodel:Relationship rdf:about="&imdb_am;MovieFavSetRelationship">
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT M
       FROM {M} rdf:type {imdb_dm:Movie}
       where M = $MovieFavElement$
       using namespace
         imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
         imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
    <am-metamodel:refersTo rdf:resource="&imdb_am;MovieMainUnit"/>
    <am-metamodel:sourceUnit rdf:resource="&imdb_am;MovieReferredSubUnit"/>
  </am-metamodel:Relationship>
<!-- MovieMenuNavigationUnit -->

<!-- SearchRefUnit -->
  <am-metamodel:SubUnit rdf:about="&imdb_am;SearchRefUnit">
    <am-metamodel:refersTo rdf:resource="&imdb_am;SearchdummyUnit"/>
  </am-metamodel:SubUnit>

  <am-metamodel:NavigationalUnit rdf:about="&imdb_am;SearchdummyUnit">
    <am-metamodel:hasRelationship
        rdf:resource="&imdb_am;SearchUnitRelationship"/>
  </am-metamodel:NavigationalUnit>

  <am-metamodel:Relationship rdf:about="&imdb_am;SearchUnitRelationship">
    <am-metamodel:refersTo rdf:resource="&imdb_am;SearchUnit"/>
  </am-metamodel:Relationship>

<!-- SearchRefUnit -->


<!-- MovieUnit -->
  <am-metamodel:NavigationalUnit rdf:about="&imdb_am;MovieUnit">
    <am-metamodel:hasAttribute rdf:resource="&imdb_am;MovieUnitTitle"/>
    <am-metamodel:hasAttribute rdf:resource="&imdb_am;MovieUnitUserRating"/>
    <am-metamodel:hasAttribute rdf:resource="&imdb_am;MovieUnitNoOfVotes"/>
    <am-metamodel:hasAttribute rdf:resource="&imdb_am;MovieUnitPhoto"/>
    <am-metamodel:hasAttribute rdf:resource="&imdb_am;MovieUnitPlot"/>
    <am-metamodel:hasAttribute rdf:resource="&imdb_am;MovieUnitGenre"/>
    <am-metamodel:hasAttribute rdf:resource="&imdb_am;MovieUnitDirector"/>
    <am-metamodel:hasAttribute rdf:resource="&imdb_am;MovieUnitWriter"/>
    <am-metamodel:hasAttribute rdf:resource="&imdb_am;MovieUnitYear"/>
    <am-metamodel:hasInput rdf:resource="&imdb_am;MovieUnitVar"/>
    <am-metamodel:hasSetUnit rdf:resource="&imdb_am;MovieUnitCastSetUnit"/>
    <am-metamodel:hasSetUnit rdf:resource="&imdb_am;MovieUnitPhotoSetUnit"/>
    <am-metamodel:onLoadQuery rdf:resource="&imdb_am;MovieUnitOnLoadQuery"/>
    <am-metamodel:hasScript rdf:resource="&imdb_am;MovieUnitPhotoScript"/>
    <am-metamodel:hasRelationship
        rdf:resource="&imdb_am;MovieUnitAddToFavorites"/>
  </am-metamodel:NavigationalUnit>

  <am-metamodel:Attribute rdf:about="&imdb_am;MovieUnitTitle">
```

```
    <rdfs:label rdf:datatype="&xsd;string">Title</rdfs:label>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
        SELECT T
        FROM {$M$} rdf:type {imdb_dm:Movie};
                   rdfs:label {T}
        using namespace
          imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
          imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
  </am-metamodel:Attribute>

  <am-metamodel:Attribute rdf:about="&imdb_am;MovieUnitUserRating">
    <rdfs:label rdf:datatype="&xsd;string">User Rating</rdfs:label>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
        SELECT V
        FROM {$M$} imdb_dm:hasUserRating {} imdb_dm:averageVote {V}
        limit 1 offset 1
        using namespace
          imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
          imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
  </am-metamodel:Attribute>

<am-metamodel:Attribute rdf:about="&imdb_am;MovieUnitNoOfVotes">
    <rdfs:label rdf:datatype="&xsd;string">NoOfVotes</rdfs:label>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
        SELECT V
        FROM {$M$} imdb_dm:hasUserRating {} imdb_dm:nrPeopleVoted {V}
        limit 1
        using namespace
          imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
          imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
  </am-metamodel:Attribute>


  <am-metamodel:Attribute rdf:about="&imdb_am;MovieUnitDirector">
    <rdfs:label rdf:datatype="&xsd;string">Directed by</rdfs:label>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
        SELECT Name
        FROM {$M$} imdb_dm:movieDirector {D} rdfs:label {Name}
        using namespace
          imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
          imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
  </am-metamodel:Attribute>

  <am-metamodel:Attribute rdf:about="&imdb_am;MovieUnitWriter">
    <rdfs:label rdf:datatype="&xsd;string">Writing Credits</rdfs:label>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT Name
       FROM {$M$} imdb_dm:movieWriter {W} imdb_dm:writingWriter {}
       rdfs:label {Name}
       using namespace
          imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
          imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
  </am-metamodel:Attribute>

  <am-metamodel:Attribute rdf:about="&imdb_am;MovieUnitPlot">
    <rdfs:label rdf:datatype="&xsd;string">Plot</rdfs:label>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT P
       FROM {$M$} imdb_dm:moviePlotOutline {P}
       using namespace
          imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
          imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
  </am-metamodel:Attribute>
```

```
<am-metamodel:Attribute rdf:about="&imdb_am;MovieUnitYear">
  <rdfs:label rdf:datatype="&xsd;string">Release date:</rdfs:label>
  <am-metamodel:hasQuery rdf:datatype="&xsd;string">
     SELECT Y
     FROM {$M$} imdb_dm:movieYear {Y}
     using namespace
       imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
       imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
  </am-metamodel:hasQuery>
</am-metamodel:Attribute>

<am-metamodel:Attribute rdf:about="&imdb_am;MovieUnitTagline">
  <rdfs:label rdf:datatype="&xsd;string">Tagline:</rdfs:label>
  <am-metamodel:hasQuery rdf:datatype="&xsd;string">
     SELECT T
     FROM {$M$} imdb_dm:movieTagline {T}
     using namespace
       imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
       imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
  </am-metamodel:hasQuery>
</am-metamodel:Attribute>

<am-metamodel:Attribute rdf:about="&imdb_am;MovieUnitGenre">
  <rdfs:label rdf:datatype="&xsd;string">Genre:</rdfs:label>
  <am-metamodel:hasQuery rdf:datatype="&xsd;string">
     SELECT G
     FROM {$M$} imdb_dm:movieGenre {G}
     using namespace
        imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
        imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
  </am-metamodel:hasQuery>
</am-metamodel:Attribute>

<am-metamodel:Attribute rdf:about="&imdb_am;MovieUnitPhoto">
  <rdfs:label rdf:datatype="&xsd;string">Photo</rdfs:label>
  <am-metamodel:hasMediaType
      rdf:datatype="&xsd;string">image/jpg</am-metamodel:hasMediaType>
  <am-metamodel:hasQuery rdf:datatype="&xsd;string">
      SELECT P
      FROM {$M$} imdb_dm:hasMainPhoto {} imdb_dm:photoURL {P}
      using namespace
         imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
         imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
  </am-metamodel:hasQuery>
</am-metamodel:Attribute>

<am-metamodel:SetUnit rdf:about="&imdb_am;MovieUnitCastSetUnit">
  <rdfs:comment rdf:datatype="&xsd;string">Cast</rdfs:comment>
  <am-metamodel:refersTo rdf:resource="&imdb_am;MovieUnitCastSetElementUnit"/>
  <am-metamodel:hasQuery rdf:datatype="&xsd;string">
     SELECT DISTINCT A
     FROM {A} rdf:type {imdb_dm:Actor};
              imdb_dm:actorFilmography {} imdb_dm:castMovie {$M$}
     using namespace
       imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
       imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>,
       imdb_cm = &lt;http://wwwis.win.tue.nl/~hera/Hera-S/imdb_cm.owl#>
  </am-metamodel:hasQuery>
</am-metamodel:SetUnit>

<am-metamodel:NavigationalUnit
     rdf:about="&imdb_am;MovieUnitCastSetElementUnit">
  <am-metamodel:hasAttribute  rdf:resource="&imdb_am;MovieActorName"/>
  <am-metamodel:hasAttribute  rdf:resource="&imdb_am;MovieActorPhoto"/>
  <am-metamodel:hasAttribute  rdf:resource="&imdb_am;MovieActorRole"/>
  <am-metamodel:hasRelationship
     rdf:resource="&imdb_am;ActorUnitRelationship"/>
</am-metamodel:NavigationalUnit>
```

```
<am-metamodel:Relationship rdf:about="&imdb_am;ActorUnitRelationship">
  <am-metamodel:hasQuery rdf:datatype="&xsd;string">
     SELECT A
     FROM {A} rdf:type {imdb_dm:Actor}
     where A = $A$
     using namespace
       imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
       imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
  </am-metamodel:hasQuery>
  <am-metamodel:refersTo rdf:resource="&imdb_am;ActorMainUnit"/>
</am-metamodel:Relationship>

<am-metamodel:Attribute rdf:about="&imdb_am;MovieActorName">
  <rdfs:label rdf:datatype="&xsd;string">ActorName</rdfs:label>
  <am-metamodel:hasQuery rdf:datatype="&xsd;string">
     SELECT DISTINCT Name
     FROM {$A$} rdfs:label {Name};
                rdf:type {imdb_dm:Actor}
     using namespace
        imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
        imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>,
        imdb_cm = &lt;http://wwwis.win.tue.nl/~hera/Hera-S/imdb_cm.owl#>
  </am-metamodel:hasQuery>
</am-metamodel:Attribute>

<am-metamodel:Attribute rdf:about="&imdb_am;MovieActorPhoto">
  <rdfs:label rdf:datatype="&xsd;string">ActorPhoto</rdfs:label>
  <am-metamodel:hasMediaType rdf:datatype="&xsd;string">
     image/jpg
  </am-metamodel:hasMediaType>
  <am-metamodel:hasQuery rdf:datatype="&xsd;string">
     SELECT DISTINCT P
     FROM  {$A$} rdf:type {imdb_dm:Actor},
           [{$A$} imdb_dm:hasMainPhoto {} imdb_dm:photoURL {P}]
     using namespace
        imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
        imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>,
        imdb_cm = &lt;http://wwwis.win.tue.nl/~hera/Hera-S/imdb_cm.owl#>
  </am-metamodel:hasQuery>
</am-metamodel:Attribute>

<am-metamodel:Attribute rdf:about="&imdb_am;MovieActorRole">
  <rdfs:label rdf:datatype="&xsd;string">ActorRole</rdfs:label>
  <am-metamodel:hasQuery rdf:datatype="&xsd;string">
     SELECT DISTINCT Cast
     FROM {$A$} rdf:type {imdb_dm:Actor};
                imdb_dm:actorFilmography {F} imdb_dm:castMovie {$M$},
                                         {F} imdb_dm:castCharacter {Cast}
     using namespace
       imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
       imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>,
       imdb_cm = &lt;http://wwwis.win.tue.nl/~hera/Hera-S/imdb_cm.owl#>
  </am-metamodel:hasQuery>
</am-metamodel:Attribute>

<am-metamodel:SetUnit rdf:about="&imdb_am;MovieUnitPhotoSetUnit">
  <rdfs:comment rdf:datatype="&xsd;string">Photo Gallery</rdfs:comment>
  <am-metamodel:hasQuery rdf:datatype="&xsd;string">
     SELECT DISTINCT P
     FROM {$M$} imdb_dm:moviePhoto {P}
     using namespace
        imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
        imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>,
        imdb_cm = &lt;http://wwwis.win.tue.nl/~hera/Hera-S/imdb_cm.owl#>
  </am-metamodel:hasQuery>
  <am-metamodel:refersTo
     rdf:resource="&imdb_am;MovieUnitPhotoSetElementUnit"/>
</am-metamodel:SetUnit>

<am-metamodel:NavigationalUnit
```

```
      rdf:about="&imdb_am;MovieUnitPhotoSetElementUnit">
    <am-metamodel:hasAttribute  rdf:resource="&imdb_am;MoviePhotoes"/>
  </am-metamodel:NavigationalUnit>

  <am-metamodel:Attribute rdf:about="&imdb_am;MoviePhotoes">
    <rdfs:label rdf:datatype="&xsd;string">MoviePhoto</rdfs:label>
    <am-metamodel:hasMediaType rdf:datatype="&xsd;string">
       image/jpg
    </am-metamodel:hasMediaType>
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
      SELECT DISTINCT PUrl
      FROM {$P$} imdb_dm:photoURL {PUrl}
      using namespace
        imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
        imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>,
        imdb_cm = &lt;http://wwwis.win.tue.nl/~hera/Hera-S/imdb_cm.owl#>
    </am-metamodel:hasQuery>
  </am-metamodel:Attribute>


  <am-metamodel:UpdateQuery rdf:about="&imdb_am;MovieUnitOnLoadQuery">
    <am-metamodel:queryBody rdf:datatype="&xsd;string">
       UPDATE {V} imdb_cm:hasNoOfViews {eval(%views%+1)}
       FROM {$U$} rdf:type {imdb_cm:RegisteredUser};
                  imdb_cm:hasMovieViews {V} imdb_cm:hasNoOfViews {views};
                 imdb_cm:hasViewsOfMovie {$M$}
       using namespace
         imdb_cm = &lt;http://wwwis.win.tue.nl/~hera/Hera-S/imdb_cm.owl#>,
         imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:queryBody>
    <am-metamodel:queryType rdf:datatype="&xsd;string">
       SERQL
    </am-metamodel:queryType>
  </am-metamodel:UpdateQuery>

  <am-metamodel:Relationship rdf:about="&imdb_am;MovieUnitAddToFavorites">
    <am-metamodel:hasQuery rdf:datatype="&xsd;string">
       SELECT M
       FROM {M} rdf:type {imdb_dm:Movie}
       where M = $M$
       using namespace
         imdb_dm = &lt;http://www.vub.ac.be/imdb-schema#>,
         imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:hasQuery>
    <am-metamodel:refersTo rdf:resource="&imdb_am;AddToFavoritesUnit"/>
  </am-metamodel:Relationship>

  <am-metamodel:Variable rdf:about="&imdb_am;MovieUnitVar">
    <am-metamodel:varName rdf:datatype="&xsd;string">M</am-metamodel:varName>
  </am-metamodel:Variable>

  <am-metamodel:Variable rdf:about="&imdb_am;movieName">
    <am-metamodel:varName rdf:datatype="&xsd;string">
      movieName
    </am-metamodel:varName>
  </am-metamodel:Variable>

  <am-metamodel:Variable rdf:about="&imdb_am;actorName">
    <am-metamodel:varName rdf:datatype="&xsd;string">
       actorName
    </am-metamodel:varName>
  </am-metamodel:Variable>
<!-- end MovieUnit -->

<!-- AddToFavoritesUnit -->
    <am-metamodel:LogicUnit rdf:about="&imdb_am;AddToFavoritesUnit">
     <am-metamodel:onLoadQuery rdf:resource="&imdb_am;AddToFavoritesQuery"/>
     <am-metamodel:hasRelationship
        rdf:resource="&imdb_am;MovieUnitResultRelationship"/>
    </am-metamodel:LogicUnit>
```

```
  <am-metamodel:UpdateQuery rdf:about="&imdb_am;AddToFavoritesQuery">
    <am-metamodel:queryBody rdf:datatype="&xsd;string">
       Construct {$U$} imdb_cm:hasFavorite {$M$}
       FROM {$U$} rdf:type {imdb_cm:RegisteredUser}
       using namespace
         imdb_cm = &lt;http://wwwis.win.tue.nl/~hera/Hera-S/imdb_cm.owl#>,
         imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:queryBody>
    <am-metamodel:queryType rdf:datatype="&xsd;string">
       SERQL
    </am-metamodel:queryType>
  </am-metamodel:UpdateQuery>

<!-- AddToFavoritesUnit -->

<!-- AddToFavoritesActorUnit -->
    <am-metamodel:LogicUnit rdf:about="&imdb_am;AddToFavoritesActorUnit">
     <am-metamodel:onLoadQuery
         rdf:resource="&imdb_am;AddToFavoritesActorQuery"/>
     <am-metamodel:hasRelationship
         rdf:resource="&imdb_am;ActorUnitResultRelationship"/>
    </am-metamodel:LogicUnit>

  <am-metamodel:UpdateQuery rdf:about="&imdb_am;AddToFavoritesActorQuery">
    <am-metamodel:queryBody rdf:datatype="&xsd;string">
       Construct {$U$} imdb_cm:hasFavoriteActor {$A$}
       FROM {$U$} rdf:type {imdb_cm:RegisteredUser}
       using namespace
         imdb_cm = &lt;http://wwwis.win.tue.nl/~hera/Hera-S/imdb_cm.owl#>,
         imdb-cont = &lt;http://www.vub.ac.be/imdb-cont#>
    </am-metamodel:queryBody>
    <am-metamodel:queryType
       rdf:datatype="&xsd;string">SERQL</am-metamodel:queryType>
  </am-metamodel:UpdateQuery>

<!-- AddToFavoritesActorUnit -->

</rdf:RDF>
```

Appendix C

UML Diagrams

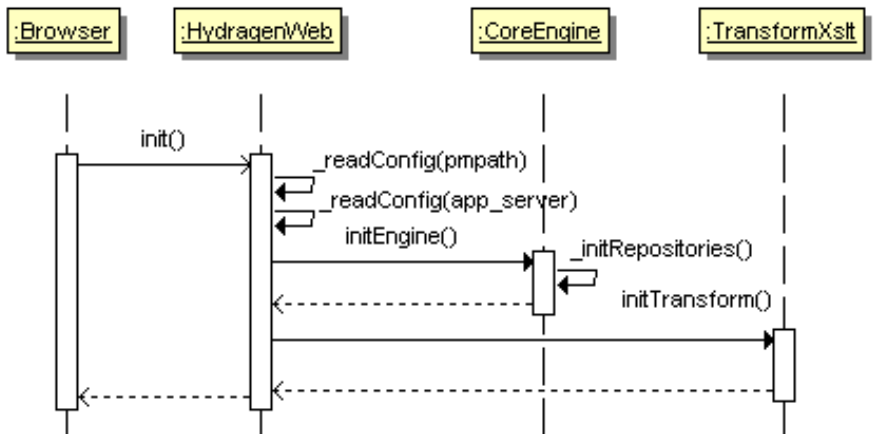

Figure C.1: Hydragen init

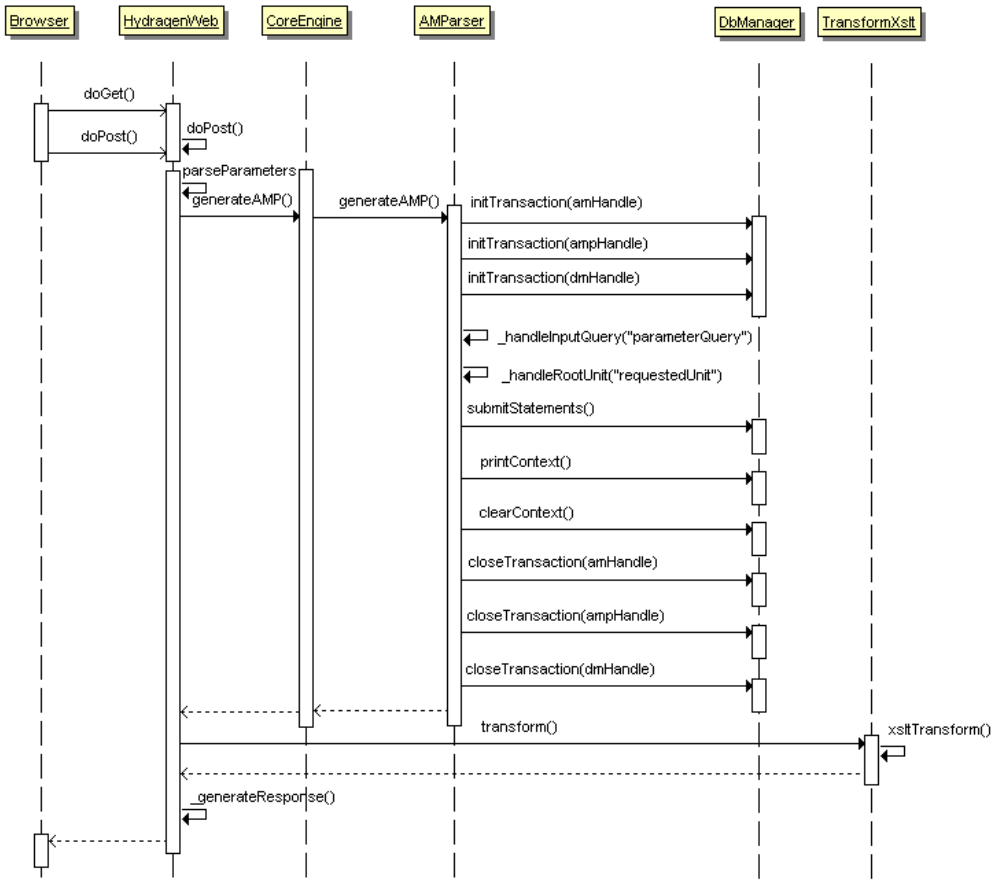

Figure C.2: Hydragen request

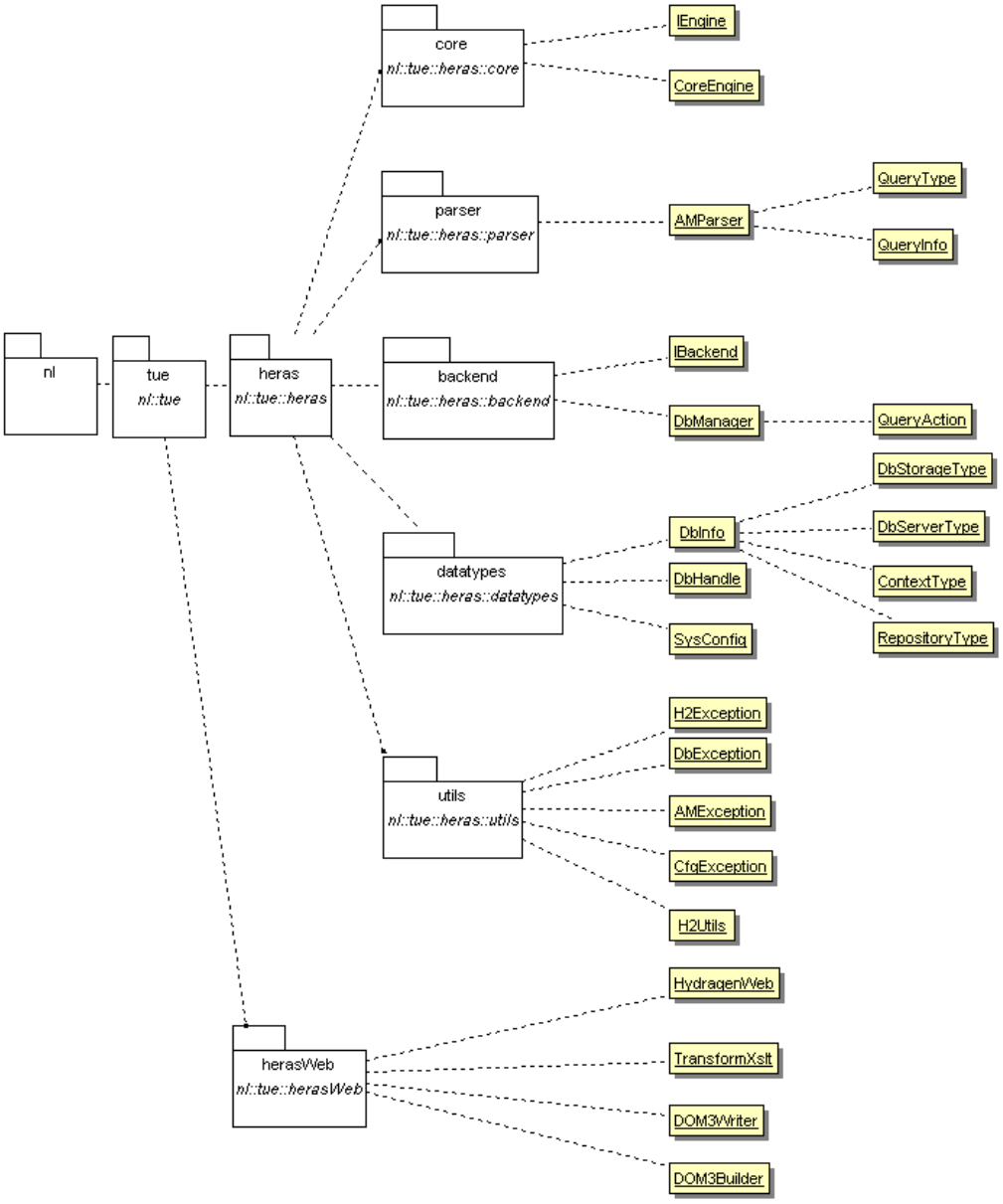

Figure C.3: Hydragen Package Diagram

List of Figures