New Review of Hypermedia and Multimedia
Vol. 15, No. 1, pp. 39 - 91

Aspect-Oriented Adaptation Specification in Web Information Systems: a Semantics-based Approach

SVEN CASTELEYN[1], WILLIAM VAN WOENSEL[1], KEES VAN DER SLUIJS[2], GEERT-JAN HOUBEN[2,3]

[1]*Department of Computer Science, Vrije Universiteit Brussel, Brussels, Belgium*

[2]*Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, Eindhoven, The Netherlands*

[3]*Software Technology Department, Delft University of Technology, Delft, The Netherlands*

By tailoring content access, presentation, and functionality to the user's location, device, personal preferences and needs, Web Information Systems have become increasingly user- and context dependent. In order to realize such adaptive behavior, Web engineers are thus faced with an additional challenge: engineering the required adaptation concerns. In this article, we present, in the context of a Web Information System design method, an adaptation engineering process that is separated from the regular Web design process. Our approach is based on the use of two key elements: (i) aspect-oriented techniques to achieve the separation of (adaptation) concerns, and (ii) the exploitation of semantic information and meta-data associated with the content, for enhanced expressivity and flexibility. By combining these key elements, we demonstrate a robust, rich, consistent and flexible way to specify adaptation in Web Information Systems.

Keywords: Web Engineering; Aspect-Orientation; Adaptation; Semantic Web

1. Introduction

The World Wide Web has become the world's main deployment platform for large hypermedia-based information systems and applications that disseminate information and services to a global audience. With the evolution from small-scale and static Websites to complex Web applications, the need for a systematic and sound approach to engineer such applications arose. In response, Web Information System (WIS) design methods aim to provide the Web developer with the necessary techniques, modeling primitives and tools to cover the complete Web design cycle, from requirements engineering to deployment and maintenance. Although many approaches have already been proposed (see related work), the rapid technological evolution of the Web, driven by advances in both hardware and software, is constantly leading to new/changing requirements and challenges in Web design. Next to challenges, these create important new opportunities as well, as we will see in this article.

One important hardware evolution shaping the future direction of the World Wide Web, and creating new requirements, lies in wireless network technologies. These finally seem to have reached the maturity required for commercial success. This is due to two key ingredients: increased bandwidth, closing up the gap with common broadband Internet speeds, and widespread availability, mainly achieved by rapid deployment and price-drop of new and superior standards by mobile telecom

operators (e.g. UMTS). Not coincidentally, advances in the capabilities of handheld and portable devices (e.g. mobile phones, PDA's, portable game-consoles) have been equally large. More economical power-usage, increased graphical capabilities and heightened processing power have turned these devices into full-fledged Web clients. As a consequence, network environments today are heterogeneous and omni-present; users are now truly accessing the Web from anywhere, with any device, and at anytime. Providing Web access in this heterogeneous setting further complicates the task of Web engineers, as users expect the WIS to exhibit some context- and user-dependency when delivering the content, according to their particular need and situation. Therefore, while adaptation and personalization requirements were already present previously in WIS design, they now demand a more prominent role. Treating adaptation engineering as a design concern in its own right, specifying and controlling it separately from the rest of the design, thus becomes indispensable.

A parallel software evolution in the World Wide Web impacting the design of WIS has been the emergence of the Semantic Web. With the gradual release of Semantic Web technologies and standards, the World Wide Web consortium (W3C)[1] has given data engineers and Web practitioners the necessary tools to provide the semantics of their data, making its meaning explicit by means of meta-data and ontologies. Today, ten years after the release of the first building blocks (i.e., XML), the Semantic Web is becoming a reality, as more and more data is becoming available in Semantic Web format, i.e. RDF to describe (meta-data about) resources, RDFS to describe the types of resources and the basic relationships between them, and OWL to explicitly represent the meaning of terms in vocabularies and further specify the relationships between them. Just to point out some examples: Wordnet, a popular English lexical database, has been available in RDF format for some time now[2]; the Open Directory database, one of the Web's largest human-edited Web directories, is also accessible in RDF[3]; scientific data is increasingly made available in Semantic Web format (see e.g. Uniprot RDF[4], a database storing protein sequences in RDF format); Wikipedia, the free online encyclopedia, can be accessed in various RDF-notations[5]; and modern Web 2.0 applications (e.g. YouTube, MySpace, Flickr) provide a wealth in user-provided meta-information in the form of tagging. The increased availability of semantic information leads to exciting new possibilities: semantics-based search engines[6], automated agents crawling and extracting relevant information, complex querying[7], (automatically) combining information from different sources[7], etc. These examples exploit Semantic Web techniques and approaches that offer opportunities for Web engineers as well. One opportunity, which we will explore in this article, lies in exploiting semantic meta-data (i.e. meta-data that is expressed using Semantic Web technology) for supporting adaptation engineering in a rich and flexible way.

Typically, an adaptation concern is spread across the entire Web application (design) (e.g. "replace all images by a descriptive text when a small screen is detected", "remove all adult-content for minors", etc), and thus cannot be localized to a particular place in the application. In regular software engineering, aspect-orientation is a proven abstraction and modularization technique to tackle such "cross-cutting" design concerns. Given the cross-cutting nature of adaptation, it is our proposition that aspect-oriented techniques can serve equally well to address adaptation concerns, and thus allow us to separate the adaptation specification from the regular Web engineering. Furthermore, the different adaptation concerns can be separated from each other, capturing each concern in an aspect. Additionally,

[1] See http://www.w3.org/
[2] See http://www.w3.org/2001/sw/BestPractices/WNET/wn-conversion.html
[3] See http://rdf.dmoz.org/
[4] http://dev.isb-sib.ch/projects/uniprot-rdf/
[5] See http://labs.systemone.at/wikipedia3
[6] E.g. http://www.swoogle.com/
[7] E.g. http://dbpedia.org/

aspect-oriented adaptation reduces the required effort when specifying adaptation, by preventing unnecessary (adaptation) code duplication and allowing for generic selection of elements relevant to the adaptation concern. The exploitation of semantic meta-data, as we will demonstrate in this article, can furthermore enhance the expressiveness of the element selection. In this article, we will thus show that by combining aspect-oriented techniques with the use of semantic meta-data, we obtain simpler yet more flexible, robust, powerful and less error-prone adaptation support, in addition to a higher degree of re-usability.

Our semantics-based aspect-oriented adaptation approach is materialized in the form of a domain-specific language, baptized SEAL. It is presented in the context of a WIS design method, Hera-S, which is perfectly suited for our aim, as it (i) naturally builds on Semantic Web data, and (ii) was conceived with adaptation in mind. To demonstrate the practicality of our proposal, we discuss how SEAL was implemented and integrated in Hydragen, an implementation generation tool for Hera-S.

This article is structured as follows. Section 2 presents related work in WIS design methods and adaptation engineering. Section 3 reviews Hera-S, the WIS design method used to illustrate our approach. Section 4 shortly elaborates on existing adaptation support in Hera-S. Section 5 explains why and how aspect-orientation is used to specify complex adaptation, based on semantic (meta)data. Section 6 discusses the design and implementation of HydraGen, the implementation generation tool for Hera-S. Section 7 discusses the realization of semantics-based aspect-oriented adaptation support by elaborating on the SEAL implementation and detailing its integration within HydraGen. Finally section 8 states conclusions.

2. Related work

This section will review related work in the key areas of this article, namely WIS design methods and their adaptation support in Web Engineering, and the use of aspect-orientation in Web design.

In literature, several proposals for WIS methods can be found, e.g.[8], OOHDM (Rossi 2007), WSDM (De Troyer, 2007), WebML (Brambilla 2007), UWE (Kraus 2007), OO-H (Garrigos 2006), Hera (Houben 2007), OOWS (Pastor 2006)). Most of these WIS design methods offer some form of adaptation support. The techniques used mostly stem from the field of *Adaptive Hypermedia*, where both techniques and methods for adaptation were identified (Brusilovsky 1996, Brusilovsky 2001). In this field, a distinction is made between adaptive presentation (i.e., content or presentation is affected) and adaptive navigation support (i.e., navigation structure or link appearance is affected), each of them sub-divided in several different technologies. One simple technique, which can be applied to implement several of these technologies, is conditional element inclusion. In this technique, adaptive access to content is achieved by adding (adaptation) conditions to the regular hypermedia design: depending on the truth-value of these conditions, some content is either shown or hidden. A representative example from the Adaptive Hypermedia field is the AHA! System (De Bra 2006), which uses conditional fragments and objects. In Web engineering, adaptation support generally adheres to this approach. In OOHDM/SHDM, conditions on navigation concepts and relationships are specified in the navigation class model (Schwabe 2004), and determine the visibility of content and navigation (links) respectively. In WebML, navigational conditions are specified as queries over the data (Ceri 1999) in a so-called profile. In WSDM, conditions associated to links in the navigation model determine the appearance of those links (De Troyer 2001). UWE, an OO-based Web engineering method, specifies adaptation conditions as OCL constraints on the conceptual model (Koch 2001). All the aforementioned methods rely on a (dedicated) engine to interpret the adaptation conditions while parsing the design models, in

[8] Citing recent publications describing the current state of the respective WIS design methods.

order to generate the desired (adapted) implementation (mostly in (X)HTML). Although condition-based approaches are relatively straightforward to apply, they also suffer some drawbacks: the specified adaptation is hard-coded and localized (i.e. one condition only applies to one particular element) and requires laborious authoring. We will go into detail on these drawbacks when discussing existing adaptation support in Hera-S, which is condition-based (see section 4).

Some methods apply (Event-)Condition-Action ((E)CA) rules to specify adaptation. In fact, AHA! deploys condition-action rules (based on events which are implicit, i.e. a page load) mainly to update the user model. In Web Engineering, OO-H employs the Personalization Rule Modeling Language (PRML) (Garrigós 2005) as a method independent language to specify adaptation. PRML supports acquisition rules to update the user model in a similar way to AHA! (yet more explicitly stating the event triggering the rule), and personalization rules to specify personalization actions. Personalization actions mostly consist of showing/hiding information fragments or links, yet also support link sorting. WSDM uses the Adaptation Specification Language (Casteleyn 2003), an ECA based rule language, to specify adaptation based on user information gathered from *all* users. The actions constitute basic transformations on the navigation model, thus altering the link structure. WebML, in the context of user-behavior aware Web Applications (Ceri 2005), applies ECA rules to achieve context-awareness. Events consist of page requests, conditions reference the user model and actions adapt page content, trigger automatic navigation (i.e. page redirects) or alter between site views or presentation styles. Rule-based approaches generally rely on a rule-engine, used in cooperation with the generation engine, to execute the adaptation rules and generate the desired (adapted) implementation. Although rule-based approaches do offer additional possibilities compared to condition-based approaches, they are slightly more complex and suffer from the fact that rules either are hard-wired to particular elements, or explicitly need to enumerate the elements to which they apply. In other words, although rule-based approaches do provide a better separation of concerns by capturing the adaptation specification in rules, the specified adaptation is, to some extent, still localized and hard-coded.

Besides preliminary work leading to this article (Casteleyn 2006, Casteleyn 2007), we came across two approaches in Web Engineering that apply aspect-orientation to specify adaptation. In the context of UWE, aspect-orientation has been used to specify some specific types of adaptive navigation (Bausmeister 2005), namely adaptive link hiding, adaptive link annotation, and adaptive link generation. However, the granularity of the considered aspects is totally different compared to the approach presented in this article: where (Bausmeister 2005) considers one particular kind of adaptation (e.g., link hiding) as an aspect, the approach described in this article allows adaptation actions constituting arbitrary transformations on the design level. Aspect specification is thus considered at a higher level of abstraction, and uses semantic information as a key element. Furthermore, an important drawback of (Bausmeister 2005) is its manual pointcut specification, i.e. each element of the pointcut requires manual annotation; in our approach, pointcut specification is done using a generic querying mechanism, which is in fact one of the main advantages of our approach. In the context of ubiquitous Web applications, WebML was extended with aspect-oriented techniques[9] (Schauerhuber 2007). This research was initiated and performed independently and in parallel to this work. In order to extend WebML with aspect-orientation, the orginal WebML language was re-modeled (with some changes) in UML, so classical aspect-oriented tools and techniques could be used. Much as in our approach, adaptation specification was separated from the regular design using aspect-orientation. However, this approach does not combine semantic information with aspect-orientation, which, as we will show in this article, provides additional strength for our approach. Also, it is not able to take into account global

[9] It should be noted that this was done by researchers external to the WebML team.

application properties when selecting elements, which, as we will also show, can help to provide for more flexible, robust and reusable adaptation aspects.

Finally, we point the interested reader to a recent survey on ubiquitous applications, containing details on most of the methods discussed above (Schwinger 2008).

3. Hera-S: an overview

Hera-S is a Web Information System (WIS) design method that combines the strength of Sesame (Broekstra 2002), a popular open source RDF framework, and the rich modeling capabilities of Hera (Houben 2007), a model-driven approach for engineering Web applications based on semantically structured data. As is the case for most existing WIS design methods, Hera-S distinguishes three design phases: domain (during which the underlying domain is modeled), navigation (during which Web application logic and navigation is modeled) and presentation (during which layout and look and feel is modeled). By clearly separating these design concerns in separate design models, the complexity of creating large Web applications is effectively reduced. Key feature of Hera-S is the fact that it is completely based on RDF technology: each design model is written using RDF(s) and/or OWL. The model implementing the domain design is called the Domain Model (DM), which represents an RDF(S) or OWL-based specification of the content data. Many model-driven engineering methods include domain design as an intrinsic part of the application design where Hera-S does not. While it is still possible in Hera-S to manually create the data and DM for each separate application, it mostly targets the reuse of existing domains and their data instantiations. Many domains and corresponding data already exist in RDF format, and in Hera-S it is possible to reuse these as a basis for the application that is being designed.

The domain data is accessed via the Sesame RDF framework and its SeRQL query language[10].
The purpose of the DM is to define the semantic structure of the content data: it expresses what needs to be known about the content over which the application will be working. The main design phase in Hera-S targets the so-called *Application Model* (AM), in which application designers can define navigable logical compositions over the RDF data, i.e. define a hypermedia-based navigation structure over the content. This navigation structure is provided in order to deliver and present the content to the user in a way that allows for a (semantically) effective access to the content. Finally, the *Presentation Model* (PM) adds the necessary details for presentation (e.g. positioning, look-and-feel). The PM in Hera-S is an overlay over the AM, which extends the navigation structure of the AM with information about presentational aspects. However, the PM is optional and Hera-S can be easily configured to use an external Presentation Engine that uses a different kind of PM, as we demonstrated in e.g. (Fiala 2004). In our current Hera-S implementation, we use XSLT stylesheets to serve as the PM.

Another distinctive characteristic of Hera-S is its explicit focus on adaptation and personalization. For this purpose, Hera-S maintains a *Context Model* (CM), which allows for user- and context-based adaptation. The CM is also expressed in RDF, and can contain references to the DM (e.g. when allowing interests in domain concepts). Context data (i.e. the CM instances) allows Hera-S to perform user- and context-based adaptation. In general, such context data can be separated into three different types. On the lowest level there is session data, which is about a specific user and his/her current situation in a specific user session. For example, session data can include information about the operating system, browser and device, the pages that have been browsed in this session, the time during which this session has been active, etc. One level higher we discern user data. User data is information that has been

[10] SeRQL can be easily exchanged for SPARQL, the W3C RDF query standard.

aggregated over several sessions, and provides an insight into the user's behavior, knowledge, preferences, etc. This data is typically used for personalization, but gives rise to the need of a user authentication system. On the highest level we discern global data, which represents context information that supersedes one single user. For example, this includes information about the total number of page views of a particular page, stereotype information, friendship relations between users, etc. Hera-S allows access to the context data by other processes, so that they can use and update this information. This allows for the context data to be accessed and manipulated by business logic software to e.g. extract popular visitation patterns, or by external user modeling frameworks for user model exchange.

An illustrative overview of the Hera-S architecture can be found in Figure 1. Fed by data from the actual data source, which conforms to the Domain Model, the Application Model is instantiated, resulting in so-called *Application Model Pages* (AMPs). Using the presentation model, the actual Web pages can be generated.

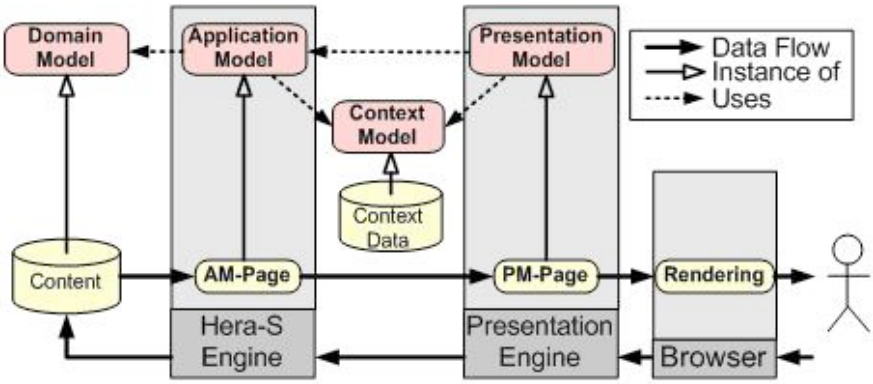

Figure 1. Hera-S architecture

To later understand the semantic-based adaptation illustrations expressed in Hera-S, let us consider some small examples that we take from a single running case: the IMDB Website[11]. This large-scale case is a copy of the actual IMDB Website, and was set up to experiment and test various aspects of Hera-S, among which the aspect-oriented adaptation support as proposed in this article. To give a small idea of its scale: the DM for our IMDB Website roughly consists of thirty concepts and over hundred (data- and object-) properties. The actual content was obtained using our dedicated crawler, which extracted a representative data set (296·233 RDF triples, consisting of 23·626 movies and 1481 persons) from the actual IMDB Website.

Here, we focus our examples on the AM, where adaptation will be specified. When exploiting the semantics of the content data to perform adaptation, we will also use the DM, but as this is simply standard RDF(S) it does not require any further clarification. Our examples are expressed using Turtle[12], a well-known, compact, and easy-to-use RDF syntax.

The central AM notions are (Navigational) Units, Attributes and Relationships. Units represent chunks of content information, and thus group those elements that semantically will be shown together to the user. Units typically correspond to a single concept in the DM, yet this is not mandatory. They can for instance refer to several domain concepts or no domain concepts at all (e.g. a fixed welcome page in an application). The elements contained in a Unit are either Attributes, Relationships or other Units

[11] http://www.imdb.com/
[12] http://www.dajobe.org/2004/01/turtle/

(called Sub-units). Attributes are literal values, typically representing strings although any other URL-referable media type is allowed (e.g. pictures, videos, html pages). Attributes mostly refer to (datatype) properties in the DM[13], and thus specify exactly which of these properties will be presented to the user. Sub-units allow Navigational Units to be aggregated inside other Units, while Relationships represent (browsable) links between units. Both elements are built upon underlying semantic relationships between concepts of the DM. There are two general types of Sub-units: 'regular' units and 'set' units. A Set-unit can lead to several units being aggregated in the Navigational Unit, while a Sub-unit only leads to one aggregated Unit.

The AM for our IMDB running case consists of over thirty units and sub-units, of which we will show two (simplified) examples. Consider the case where we want to show a movie (by its title and plot) and its cast (their names, pictures and links to their biographies). In Turtle notation, the AM includes:

```
:MovieUnit a ams:NavigationalUnit ;
  ams:hasInput [ a ams:Variable ;
                 ams:varName "M";
                 ams:varType imdb:Movie] ;

  ams:hasAttribute [
    rdfs:label "Title" ;
    ams:hasQuery
      "SELECT T FROM {$M} rdf:type {imdb:Movie};
                        rdfs:label {T}"] ;

  ams:hasAttribute [
    rdfs:label "Plot" ;
    ams:hasQuery
      "SELECT P FROM {$M} rdf:type {imdb:Movie};
                        imdb:moviePlotOutline {P}"] ;

  ams:hasSetUnit [
    rdfs:label "Cast" ;
    ams:refersTo :ActorUnit ;
    ams:hasQuery
      "SELECT A FROM {$M} rdf:type {imdb:Movie};
             imdb:movieActor {A}"
  ] .

:ActorUnit a ams:NavigationalUnit ;
  ams:hasInput [ a ams:Variable ;
                 ams:varName "A";
                 ams:varType imdb:Actor] ;

  ams:hasAttribute [
    rdfs:label "Name" ;
    ams:hasQuery
      "SELECT N FROM {$A} rdf:type {imdb:Actor};
                        rdfs:label {N}"] ;
```

[13] More specifically, a datatype property of the concept that corresponds to the Unit in which the attribute appears.

```
  ams:hasAttribute [
    rdfs:label "Photo" ;
    ams:hasQuery
      "SELECT Ph FROM {$A} rdf:type {imdb:Actor};
                       imdb:hasMainPhoto {Ph}"] ;

  ams:hasRelationship [
    rdfs:label "Actor Bio Page" ;
    ams:refersTo :ActorBioUnit ;
    ams:hasQuery
      "SELECT B
       FROM {$A} rdf:type {imdb:Actor};
                 imdb:actorBio {B}"
] .
```

The example consists of two units: MovieUnit and ActorUnit (and a reference to the ActorBioUnit, which we will not discuss further). Both units contain some elements and represent a particular grouping of information. The MovieUnit contains two attributes, one representing the title of a movie, the other representing the plot of a movie. Finally, the MovieUnit also contains a set-unit, representing the cast for that movie. Note that the set-unit "refersTo" the ActorUnit, which specifies what exactly to show for every Actor. We chose here to use a set-unit, as opposed to a navigational relationship, which will result in the final Web page showing the cast information inside the same page as the movie (instead of links to pages representing the actors). The ActorUnit contains attributes that show the actor's name and photo, and a relationship to the 'Actor Bio Page'. The latter represents a browsable element, which in this case links to a page that shows the biography for this actor.

As already mentioned, a unit will mostly correspond to (a) specific domain concept(s); in that case, one or several content elements are needed in order to instantiate the unit. For example, in case of the Movie unit, it must be known what specific movie is represented by the unit, in order to be able to retrieve the correct title, plot and cast members. The output of queries associated with sub-units and relationships are used for this purpose. For example, in the Movie unit the output of the SeRQL query for the set-unit element will provide a number of actor content elements, each of which will be used to instantiate an Actor unit. This passing of information is done via variables (denoted by the ams:hasInput property), which are assigned the output values of those queries. Other queries inside the unit can reference these variable values, in order to retrieve information specific to the unit's current content element: e.g. in case of the ActorUnit, this allows the name and photo of the specific actor to be retrieved. Variables can also be instantiated within a unit itself, where they are either bound to a constant value or to the output of a specific SeRQL query.

Next to the above basic constructs, Hera-S also supports others. An important construct is the update-query. These queries are defined by the designer, and define an update that needs to be performed (at runtime) upon the context and user data. Whenever the AM element (e.g. unit, attribute, relationship) containing the update query is instantiated (or link clicked in case of a relationship), the required update is performed (see section 6.3 for realization of this functionality). This mechanism is vital for adaptation, and allows among others logging of user activity (e.g., amounts of page loads, link clicks). Other constructs include forms, scripts, (guided) tours, Web services and frame-based navigation. For details on these elements we refer to (Van der Sluijs 2006, Houben 2007).

4. Existing Adaptation Engineering in Hera-S

As already mentioned in the related work section, existing adaptation support in Hera-S is based on conditional element inclusion. The conditions are integrated in the SeRQL expressions underlying every element of the Application Model, making the instantiation of the particular AM element dynamic. If the condition references the Context Model (denoted by the "cm:" prefix), it thus expresses user- or context-dependency, and so the resulting Web application becomes adaptive to the current user or context. Continuing the example of section 3, consider a DirectorUnit, but imagine we don't want to display the director's picture when the user is using a small-screen pda-like device, and furthermore, directed movies should only be shown when the director is in the user's favorite director list. The DirectorUnit would then look as follows (for clarity, the adaptation conditions that were added are put in bold; non-altered parts were omitted):

```
:DirectorUnit a ams:NavigationUnit ;
  ...
  ams:hasAttribute [
    rdfs:label "Photo" ;
    ams:hasQuery
      "SELECT Ph
       FROM {$D} rdf:type {imdb:Director};
                 imdb:hasMainPhoto {Ph},
                 {} rdf:type {cm:CurrentUser};
                 cm:userDevice {Dev}
                 WHERE NOT Dev LIKE 'pda'"
  ] ;

  ams:hasRelationship [
    rdfs:label "Movies directed" ;
    ams:refersTo :MovieUnit ;
    ams:hasQuery
      "SELECT M
       FROM {$D} rdf:type {imdb:Director};
       imdb:directorFilmography {M} imdb:hasAgeRating {}
       imdb:minAllowedAge {Min},
       {cm:CurrentUser} cm:age {Age}
       WHERE Age >= Min"
  ] .
```

It shows that using SeRQL, arbitrary user or context conditions can thus be written into any regular selection query present in the AM. While SeRQL is both versatile and expressive and therefore allows arbitrary and powerful personalization, inherently the way in which adaptation is specified in the global Web application design suffers from some drawbacks:

- *Adaptation specification is localized*: In the above example, we restricted the display of pictures from directors. However, adaptation is typically not localized: e.g. for small-screen users it makes more sense to eliminate all pictures from their application, instead of only director pictures. While certainly possible with the above approach, it would require not only restricting pictures of directors (in DirectorUnits), but instead eliminating all pictures, wherever their appearance is specified in the Application Model. This forces the designer to (manually) identify

all attributes of units in the AM where pictures are shown, and subsequently alter the corresponding queries in all these units.

- *Adaptation specification is hard-coded*: as mentioned in the example of the previous bullet, the designer needs to identify (over the entire AM) which queries will return pictures, and subsequently alter these queries. The adaptation specification is thus hard-coded and not re-usable. Furthermore, when extending the AM later on (for example, by adding songs or games to the Web site) and possibly introducing new pictures, these pictures would still be shown to the user, unless adaptation conditions are also (manually) added for these new parts of the AM.
- *Semantic information is ignored*: valuable semantic information, present in the Domain Model, is ignored. For instance, in the previous example of omitting pictures for small screen users, the semantic (type) information for media elements (present in the Domain Model) could be exploited to automatically select all pictures. Instead, it is left to the designer to interpret which queries involve pictures.
- *Global (structural) properties are not usable*: because adaptation conditions are limited to one particular element (i.e. integrated in a particular query), they cannot express restrictions based on several elements or, more general, on global properties of the Application Model. For example, expressing that pictures can be shown to small screen users, but only on pages where no other pictures appear, is currently impossible.

The above problems leave room for human error, with possible inconsistencies in the adaptation behavior as a result. Additionally, adaptation engineering is also totally intertwined with the (classical) Web engineering (concerns), complicating the overall design.

A more systematic and high-level approach, treating adaptation as a design concern in its own right, is thus called for. In the next section, we address this issue by presenting an aspect-oriented and semantic-based approach to adaptation specification. Our approach effectively separates adaptation engineering from (regular) Web engineering, and tackles the fact that adaptation concerns are typically spread over the entire Web application. In order to add strength, flexibility and robustness to the adaptation specification, we exploit the valuable semantic information typically present in the underlying domain, in addition to (global) structural properties of the Web application.

5. Aspect-Oriented Adaptation Engineering

As established in the previous section, adaptation design is typically spread all over the Web application design. A similar observation was made in the programming community. When studying certain design concerns, it gradually became clear that some concerns cannot be localized to one particular function or module. A clear example is logging, the code of which is typically spread over the entire application code. Such a concern is called cross-cutting. The answer of the programming community to address such a concern while maintaining good abstraction and modularization is aspect-orientation (Kiczales 1997): instead of duplicating the required logging code in different places in the application code, it is specified as an aspect. An aspect consists of a pointcut and an advice. The pointcut performs a query over the programming code, selecting exactly those locations where the cross-cutting concern comes into play. The advice consequently specifies which action should be taken whenever this cross-cutting concern is detected (i.e. which piece of code should be 'injected'). Exactly the same paradigm can be applied for adaptation specification: some (adaptation) action(s), i.e. the advice, is typically required to be in multiple places in the Web application, i.e. the pointcut. The pointcut will thus consist of a query over the AM, selecting exactly those elements which require an action to be

taken for adaptation. The action itself will typically consist of injecting adaptation-conditions to particular places selected by the pointcut. However, we will not limit ourselves to only adding adaptation-conditions. In fact, as we will see, an adaptive action will consist of a arbitrary transformation applied to the AM elements selected in the poincut, and will thus also allow adding, deleting or replacing elements, based on a certain (context- or user-dependent) condition. In the next two subsections, we will discuss pointcuts and advices in detail in this adaptation engineering perspective.

We devised our own Aspect Language, baptized SEAL (**Se**mantics-based Aspect-Oriented **A**daptation **L**anguage), which is custom-made to provide adaptation support in the context of Hera-S. Such languages, specifically aimed at supporting certain tasks in a specific domain, are called *domain specific languages* (Van Deursen 2000). Their advantages are well-documented, and clearly motivate our choice for a newly designed language here, as opposed to using a general-purpose language. Most important benefits are reduced complexity (constructs take into account the peculiarities of Hera-S models, and thus allow easy and quick access and handling of these models) and ease of use (domain experts may readily understand and use our domain-specific language, and do not need to learn a more complex general-purpose language). In this section, we systematically explain all the SEAL language constructs and their semantics, and illustrate them with examples. Interested readers are referred to http://wise.vub.ac.be/downloads/research/seal/SEALBNF.pdf for the full SEAL syntax in BNF notation.

5.1 Pointcuts

Pointcut expressions select exactly those elements from the Application Model where adaptation concerns need to be applied. The basic pointcut statement selects *all* AM elements. All subsequent language constructs restrict (condition) this selection in some way:

Type restriction: Using the "type" construct, selection of elements is restricted to those of a certain type. The most important available Hera-S element types are (for a full list, see appendix A): "unit", "subunit", "set", "form", "nav" "attribute", "relationship", "label", "query", "tour", "target", "source". Elements of different types may be selected by specifying all desired types, separated by a "," (denoting disjunction, see the second example below)[14]. Typical examples include:

- ; (selects all AM elements)
- type unit, subunit; (selects all units and sub-units)
- type attribute; (selects all attributes)

Conditions: Next to restricting the element selection according to type, it may also be restricted according to name, value (a property of an AM-element should have a certain value; see the fourth example where it is specified that the label of an attribute should equal a certain value using the *hasLabel* keyword), aggregation (an element *contains* another element, or is *containedIn* another element), or aggregate function (using a numerical condition on the amount of contained/containedIn elements, specified using *count*). The navigational structure of the AM can also be exploited to restrict element selection: for relationships, restrictions on source and target may be specified (using *from* and *to*), while units can also be selected based on the navigational relations between them (using *navigateFrom* and *navigateTo*). Where appropriate, string pattern-matching is allowed when conditioning values (see the first example below). Logical expressions may be negated or combined using logical conjunction or disjunction; parentheses are used to alter default logical operator precedence. Typical examples include:

[14] In fact, this is syntactic sugar for "type unit or type subunit" which we'll see further on.

- type unit or type subunit; (select all units and subunits; this in fact is equivalent to a previous example, namely "type unit, subunit")
- type unit and hasName "movie*"; (selects all units that have a name that starts with "movie")
- type subunit and contains (2 < count(type attribute) < 5); (selects all sub-units which have between 2 and 5 attributes specified)
- type attribute and containedIn (type unit and contains (type attribute and hasLabel "title")); (selects all attributes contained in a unit, which has an attribute labeled "title")
- type relationship and from (type unit and contains (count(type relationship) > 3); (select all relations which originate from a unit containing more than 3 relationships)
- type unit and navigateTo (type unit and contains (type tour)); (selects all units containing a(ny) relationschip that *navigates to* a unit containing a guided tour)
- type unit and navigateFrom (type unit and hasName "*movie*"); (selects all units to which a(ny) relation points to, *navigating from* a unit with a name containing "movie")

Native calls: SEAL also supports calls to the native underlying query language (SeRQL) to retrieve AM elements. This support has been provided so that the expert user can still specify intricate queries that exploit the full expressiveness of SeRQL (and of which the semantics are not expressible in SEAL). Such queries can be specified by using the <SeRQL> keyword at the beginning of a pointcut specification. A trivial example includes:

- <SeRQL> "SELECT V4 FROM {U} ams:hasAttribute {A} rdfs:label {L} WHERE L LIKE \"title\""; (selects all attributes with a label "title")

The <SeRQL> construct can also be used to query the Domain Model, thereby exploiting its semantic information; we discuss this next.

Using semantic information: Semantic information from the Domain Model can be exploited to retrieve relevant elements from the AM, using SeRQL as the native query language to access the Domain Model. For example, to select all subunits whose corresponding domain concept (e.g. person, movie) is connected via a property to an age rating[15]:

- type subunit and hasInput in [<SeRQL> SELECT I FROM {imdb:hasAgeRating} rdfs:domain {I}];

This condition selects subunits with an input variable type that appears in the domain of the imdb:hasAgeRating property. Note that this pointcut effectively exploits domain knowledge, and alleviates the burden of the adaptation engineer to (manually) find out which domain concepts have an age rating specified. Such a pointcut cannot be specified using AM information alone, unless the designer explicitly made sure the AM rigorously "marks" the places to be selected in the pointcut. Naming conventions such as adding a label named "age-rating" could for example do this. In that case, the following pointcut would achieve the desired result:

- type subunit and contains (type attribute and hasLabel "age-rating");

In practice however, it seemed that using SeRQL queries to extract semantic information was cumbersome. In fact, our example query will only work satisfactorily if the domain contains only one element. This is often the case, but unfortunately, it is impossible to know beforehand. In OWL DL, unions of classes can be used to specify domains or ranges of properties; for example, to specify that the

[15] In SEAL, square brackets are used to enclose queries over the DM.

property “imdb:hasAgeRating” can have either a Movie *or* a Game as its domain, a union containing both these classes has to be defined as the property’s domain. Because the elements of a union are enumerated using an RDF collection, the above query will in fact return the collection (instead of the desired concepts constituting the domain). Specifying a SeRQL query retrieving the concept constituting the domain for such a property domain or range becomes very difficult, if not impossible:

- SELECT I FROM {imdb:hasAgeRating} rdfs:domain {} owl:unionOf {} rdf:first {I}

In this example SeRQL query, only the first element of the collection is considered; the developer has to make a separate query to check all elements from the collection. There is no known construct (or method) in SeRQL that allows this to be done in one query, or to even know the number of elements beforehand. Because we will be frequently querying the domain and range of properties in our DM queries, it was decided to alleviate this problem by developing a (compact) extension to SEAL that allows the referencing of the domain and range of a property in a more concise and high-level way.

The syntax of this language extension roughly corresponds to that of a simple SeRQL expression. Like in SeRQL, branching (i.e. specifying multiple properties of a single subject) and chaining (i.e. using the object of a triple directly as the subject of another triple) of nodes is possible, while only typed variables (and the variable in the SELECT clause) and URI references may occur in a path expression. Some basic inferencing is also supported: subclasses and subproperties are automatically taken into account when querying the domain or range of a property.

We thus write down the above example in our SEAL extension as follows:

- type subunit and hasInput in [SELECT node1 FROM {node1} {imdb:hasAgeRating} {node2}]

In this example, all subunits are returned of which the input variable type corresponds to *one of the* domain classes of the “imdb:hasAgeRating” property (in case a union was used to specify multiple domains for the property).

As was the case for the AM, next to this specific SEAL extension, arbitrary SeRQL queries can still be used to perform arbitrary queries over the DM.

5.2 Advices

Advices specify exactly what needs to be done to the element(s) selected in the pointcut. SEAL supports adding conditions, as is commonly done in many approaches, yet also allows arbitrary transformations of (elements of) the AM. The following constructs are available to specify advices:

Adding conditions: Conditioning elements is the common way of adapting. Conditions (condition expressions) typically reference the context data to express a kind of context or user-dependency. In analogy with Hera-S notational conventions (see section 3), referencing the user’s context is done using the “cm:”-prefix; more specifically, the current user node is used as a starting point to navigate through the context data. Analogously, referencing the domain data is done using the namespace-prefix from the Domain Model (e.g. “imdb:”); in this case, the domain resource(s) corresponding to the element(s) selected in the pointcut is used as a starting point for navigation. Navigation through the data can be done using the “.”-operator: for example, imdb:hasAgeRating.minAllowedAge returns the minimum age of a domain resource corresponding to[16] an (AM) element that was selected in the pointcut. Note that by just providing the CM namespace prefix (e.g. “cm”), the user (node) can be referenced directly. Using

[16] I.e. obtained by executing the SeRQL query underlying the AM element

"this" in the pointcut allows to explicitly reference the elements selected in the pointcut. Both are illustrated in the third example below.

As usual, condition expressions can be combined using logical conjunction, disjunction or negation, and can have parentheses altering the standard operator precedence. The semantics are such that depending on the truth value of the condition, the particular element is then shown or not. In adaptation engineering the most common practice is to include (add) conditions when a new adaptation concern is considered. Typical examples include:

- ADD CONDITION cm:age >= 18; (adds a condition to the elements selected in the pointcut denoting that the age of the user should be 18 or above, i.e. he/she is not a minor)
- ADD CONDITION imdb:hasAgeRating.minAllowedAge <= cm:age; (adds a condition to the elements selected in the pointcut which specifies that the age of the current user should be higher than the minimum allowed age for these elements).

Note that in the left part of the above example, navigation starts from the elements selected in the pointcut, while the right part specifies navigation in the context data (denoted by the "cm:" prefix) for the current user. We will elaborate further on this example in section 5.3. The observant reader will have noticed that, in some cases, this notation will be insufficient. For example, consider the following case where we do not want to show elements for which the current user has given a low rating. Using only the constructs introduced above, this would result in:

- ADD CONDITION cm:givenRating.about != this or cm:givenRating.rating > 5; (adds a condition to the elements selected in the pointcut, stating that either the user has not yet given a rating to this resource, or that the user gave a rating over 5)

It can be seen that the above condition is flawed, because it states that the user either never rated the resource or that a(ny) rating over five was given: the rating could belong to a completely different resource! Therefore, it has to be checked whether the rating that is being checked actually belongs to the resource in question (i.e. resource returned by the query). In order to combine several checks in one navigational path, we use the ";" notation:

- ADD CONDITION not cm:givenRating.about = this; rating < 5; (this condition states that in case the user has entered a rating for the resource, that rating should not be below 5)

Adding/deleting elements: (New) elements can be added to the elements selected in the pointcut if a certain condition is fulfilled, or existing elements selected in the pointcut can be deleted. When adding elements, plain RDF(S) can be added (the designer is responsible for validity), or it is possible to use ADD *something* statements where *something* is any of the AM elements. The semantics is such that these advices need to be performed at runtime upon page-request (see next section). Typical examples include:

- if (cm:age < 18) { DELETE }; (simply deletes the elements selected in the pointcut if the current user is a minor, i.e. cm:age < 18)
- if (cm:bandwith >= 1000) {
  ADD attribute containing hasLabel "Trailer", hasQuery "SELECT T FROM {$variable} rdf:type {imdb:Movie}; imdb:movieTrailer {T}";
  }; (adds, to the element(s) selected in the pointcut (movies), an AM-attribute showing the trailer, with the label "Trailer" and the corresponding query, if the user's bandwidth is above 1000 Kbps)

Replacing elements: (Parts of) existing elements selected in the pointcut can be replaced, if a certain condition is fulfilled. Only the explicitly specified parts of the elements are replaced; parts which do not appear in the replace-statement are simply copied. As with the pointcut expressions, pattern matching symbols may be used to match (part of) the element to be replaced (see the first example below). Typical examples include:

- if (cm:userDevice="pda") {
  REPLACE hasRefersTo value "Big*Unit" BY hasRefersTo
  value "Small*Unit";
  }; (if the user uses a pda, let relationships/sub-units refer to a smaller version of the unit)
- if (cm:userDevice = "pda") {
  REPLACE hasSubunit BY hasRelationship;
  }; (replaces the hasSubUnit elements by hasRelationship; all attributes of the particular sub-units are left unchanged)

Combining multiple actions: An arbitrary number of adaptation actions and conditions can be combined in a single aspect advice, as to avoid having to specify multiple aspects with the same pointcut. This is illustrated by the final example in the following subsection.

5.3 Adaptation Aspects: Examples

With pointcuts and advices described, we can now consider some examples of adaptation aspects which were specified for our running case, the IMDB Website. For each example, we will first state the adaptation requirement, and subsequently formulate the adaptation aspect realizing this requirement, followed by a small explanation. We will gradually show the strength of our approach, and illustrate and motivate how we benefit from our aspect-oriented approach, and how we exploit semantic information and structural properties of the AM when specifying adaptation aspects. We start off with a simple adaptation requirement, affecting only one particular element in the design:

Adaptation Requirement: for users that specified they do not want any spoilers, don't show the plot outline for a movie.

Adaptation Aspect:
POINTCUT: hasLabel "Plot" and containedIn (type unit and hasName "MovieUnit");
ADVICE: ADD CONDITION cm:spoilerInfo = true;

The pointcut selects the AM-element (in our case an attribute) which is labeled "Plot" and is contained in the "MovieUnit" unit. The advice adds the relevant condition to only show this "plot" attribute if the user specified he/she doesn't mind spoilers (i.e. cm:spoilerInfo = true). This first example is somewhat naive, and corresponds to typical condition-based approaches: the desired adaptive behavior is specified on one particular attribute from a specific unit. This adaptation specification does not use the full potential of SEAL: it is still localized and hard-coded. Indeed, imagine there is another movie-unit present in the AM (e.g. an elaborated version), which also shows the plot. In this case, another aspect specification would be required to also restrict visibility of this plot information, similar to typical condition-based approaches which require manual specification of conditions on all affected elements. The only advantage we have gained here is the fact that our adaptation specification is separated from the (regular) Web design; for the rest, the same drawbacks as for condition-based approach exist.

Let's now turn our attention to a more advanced example, demonstrating a cross-cutting adaptation concern:

Adaptation Requirement: restrict navigation for non-registered users to top-level links.

Adaptation Aspect:
POINTCUT: type relationship and from (type subunit) and to (type unit);
ADVICE: ADD CONDITION cm:isRegistered = true;

This pointcut selects all relationships, i.e. links, which originate from a(ny) sub-unit and target a(ny) unit. The advice indicates to add to these relationships the condition that the (current) user should be registered. Thus, the above adaptation aspect will present the non-registered user with a restricted view on the Web application: only top-level links (i.e. those appearing in units) will be shown, any link that originates from a sub-unit and targets a top-level unit, and thus typically presents an elaborated view on the particular concepts represented in the sub-unit, will be hidden. This adaptation concern is clearly cross-cutting: it is not localized, yet spread over the entire AM. In our IMDB Website, execution of the advice affected fifteen sub-units linking to top-level units. Note that this adaptation aspect exploits the structural (i.e. navigational) properties of the AM, yet is perfectly re-usable over (different) Web applications, as the adaptation specified is in no way hard-coded to the current AM.

The previous example basically restricts the visibility of (certain) links according to a specific property of the user, as stored in the Context Model. Often, a slight variation of the previous adaptation requirement occurs: one does see the links, but clicking them transfers the user to a registration page. This adaptation requirement is depicted below.

Adaptation Requirement: restrict navigation for non-registered users to top-level links by transferring him/her to a registration page.

Adaptation Aspect:
POINTCUT: type relationship and from (type subunit) and to (type unit);
ADVICE: if (not cm:isRegistered) {
REPLACE hasTarget value "*" BY hasTarget value
"am:RegistrationUnit";
};

The pointcut remains unchanged, selecting all relationships originating from a sub-unit, and targeting a unit. However, the advice doesn't simply add a condition to the selected relationships, as in the previous example. The actual adaptation that is performed is thus not filtering, as it typically done in condition-based approaches. Instead, the advice replaces, if the user is not registered, the value of the 'hasTarget' attribute (whatever it was) to the "am:RegistrationUnit" unit, causing all selected relationships to be redirected to the registration unit. In this way, the elements of the AM are actually (conditionally) altered, completely changing their behavior.

In the next example, we demonstrate how SEAL allows exploiting meta-data present in the Domain Model to perform cross-cutting adaptation:

Adaptation Requirement: don't show any age-restricted information to minors.

Adaptation Aspect:
POINTCUT: type subunit and hasInput in [SELECT node1 FROM {node1} {imdb:hasAgeRating} {node2}]

ADVICE: ADD CONDITION
imdb:hasAgeRating.minAllowedAge <= cm:age;

The pointcut selects all sub-units[17] that have as input a certain concept which has an "imdb:hasAgeRating" property specified in the Domain Model (the latter part is represented by the native SeRQL expression). The advice adds a condition to these sub-units, denoting that they should only be shown if the age of the user (specified in the context data) is higher than the minimum allowed age (specified in the domain data) for that resource. Note that in this example, no specific (AM) elements are specified in the pointcut. In other words, at specification time, it is not known which elements will or will not have an 'hasAgeRating' property. Only at runtime will it be determined which elements correspond to concepts that have an 'hasAgeRating' property specified in the Domain Model, and subsequently the corresponding elements will be selected from the AM. This clearly illustrates that the burden of identifying the place(s) in the AM where a certain adaptation needs to be performed is alleviated from the application engineer. Instead, these places are specified in a declarative way, using semantic meta-data present in the Domain Model. This adaptation aspect is thus not hard-coded, and actually quite robust: even when (later on) adding new resources to the Web application (imagine for example that IMBD decides to add songs to its collection), and therefore adding new units and/or sub-units describing these resources, the adaptation aspect will subsequently also identify these new resources and their corresponding sub-units, and restrict their access/visibility accordingly, resulting in consistent adaptation throughout the application (even when the application changes). Note that we used the SEAL extension to query the DM in the above example; if we had used a standard SeRQL query, the above aspect would not work in case multiple domains were specified (in disjunction) for the "imdb:hasAgeRating" property (see section 5.1: Using semantic information). In our IMDB running case, two concepts have a 'hasAgeRating' property specified: imdb:Movie and imdb:Game, which lead to six sub-units being identified for adaptation.

Finally, in the last example we will perform some adaptation based on a complex, global property of the Application Model, and combine multiple advice actions and conditions in a single aspect:

Adaptation Requirement: for pages with a lot of in-page information specified, replace this information by links which point to dedicated pages showing this information, in case the user is using a pda. Furthermore, target versions of these units corresponding to the user's screen size, and also notify the user that adaptation to his/her device has occurred.

Adaptation Aspect:

```
POINTCUT: type subunit and contains (count (type attribute) >= 5) ;
ADVICE: if (cm:userDevice.type = "pda") {
  REPLACE hasSubUnit BY hasRelationship;
  ADD attribute containing label "Adjusted to device",
  query "SELECT L FROM {$V} rdf:type {imdb:PDAImage};
  imdb:location {L}";
  if (cm:userDevice.screenSize < 100) {
    REPLACE hasTarget value "Big*Unit" BY hasTarget value
    "VerySmall*Unit";
    } else {
```

[17] Note that we actually also need to hide relationships pointing to age-restricted information, instead of only sub-units showing it. We have omitted relationships from the current example for clarity, but the desired behavior could easily be achieved by adding, in disjunction, an expression in the pointcut to also select these relationships (similar to the one selecting sub-units).

```
    REPLACE hasTarget value "Big*Unit" BY hasTarget value
    "Small*Unit";
  }
};
```

The pointcut selects all sub-units which have five or more attributes (i.e. "large" sub-units). Firstly, the advice replaces these sub-units (actually their in-page occurrence) by relationships to the corresponding units, to avoid crowding small screens. In this way, users will only see information directly relevant for the current page, and be presented with links to related information instead of seeing it in-page. Secondly, the fact that adaptation to the user's device has occurred is indicated by adding an attribute that displays an image of a pda-device. Finally, the targets of the relationships are replaced by smaller versions of the target; if the screen is very small, even smaller versions will be targeted. In our IMDB Website, five occurrences of sub-units containing more than five attributes were retrieved (representing both imdb:Person and imdb:Movie twice, and imdb:Theater once). Note that this adaptation aspect exploits characteristics of the entire AM, not just one single element, in this case the amount of attributes. We stress that this kind of behavior cannot be reached by current approaches which simply rely on specifying adaptation (conditions) to a single element. Furthermore note that, as in the third example, the actual adaptation that is performed is not filtering (as is commonly done), but instead alters AM elements.

6. Hera-S code generation: Hydragen

We implemented an engine called Hydragen, which is able to generate the actual Web application code based on these models. Hydragen is composed of two main components: HydragenCore and HydragenWeb. HydragenCore provides the Hera-S engine functionality. This means that given the AM, together with the content and context data, HydragenCore can instantiate the Application Model with the actual data and thus create so-called Application Model Pages (AMPs). Note that the content of AMPs thus depends on the current state of the content and context data, which makes them volatile across page requests. Therefore, HydrogenCore generates a new AMP for every request for a Navigational Unit. HydragenWeb is a Java Servlet, which basically acts as the Presentation Engine in the Hera-S framework. HydragenWeb translates user actions (e.g. clicking a link, submitting a form, etc) into requests to HydragenCore, and subsequently transforms the generated AMP into a Web page response. We currently implemented HydragenWeb so that it translates AMPs into HTML, even though it is no problem to also build translations to formats like SMIL or WML, as we showed in (Houben 2007). The translation of AMPs to HTML pages is done by using XSLT stylesheets. HydragenWeb is a relatively straightforward implementation of a presentation engine for Hera, but it is also possible to use more advanced systems such as AMACONT (Fiala 2003, Fiala 2004).

6.1 Deployment

HydragenWeb is deployed as a Web application in a servlet container, such as Apache Tomcat Server. Hydragen allows for the models and instance data to be stored on different physical locations, using the Sesame Server and Sesame Web-client. This is particularly useful in case the actual content data is not under Hydragen's direct control (e.g. in case an external repository provides the content data), and for parallelization purposes (e.g. for scalability reasons). These remote Sesame repositories are accessed by using the server URL 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. The path to the Hydragen setup file itself must be indicated in the Web application configuration file of HydragenWeb (i.e. web.xml). Sesame allows setting up several properties of the repository such as the use of an inference layer, type of storage (in-memory, native, or RDBMS), persistence, etc.

Hydragen uses Sesame as an RDF server for DM, CM and RDF content. For the AM, it uses Sesame's in-memory repository for performance reasons (as the AM model is typically relatively small in size, it is unlikely that this will lead to scalability issues).

6.2 Implementation

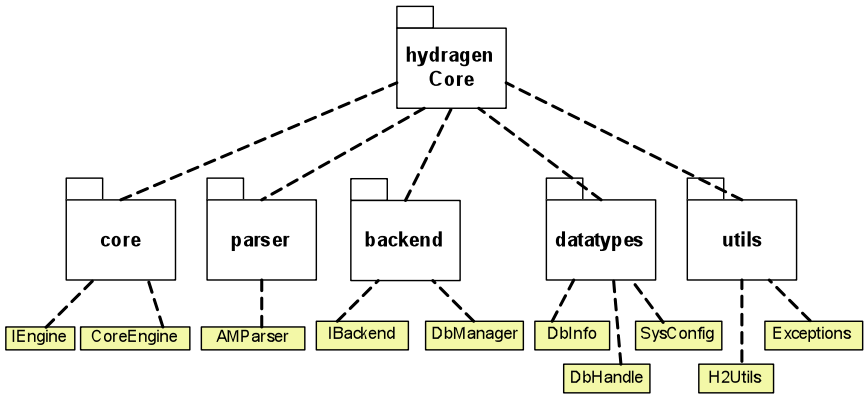

Figure 1. HydragenCore package diagram

As illustrated in Figure 1, HydragenCore consists of the following packages:

- *nl.tue.heras.core*: This package contains the main controller class (namely CoreEngine), which is responsible for initialization of the Hydragen engine and handling of requests for a Navigational Unit. The CoreEngine class implements the interface defined by IEngine. It is also responsible for checking the consistency of the data with the corresponding models.
- *nl.tue.heras.parser*: The parser package contains the Application Model parser (i.e. AMParser), which is responsible for generating Application Model Pages based on the AM and content/context data. The AMParser uses SeRQL queries to identify and extract various AM elements, process the SeRQL queries associated with *AM eleme*nts, and generate RDF statements based on the acquired results. AMParser uses the DBManager class to access the data (and model) repositories.
- *nl.tue.heras.backend*: The backend package contains classes such as DBManager, which manages all accesses to the RDF repositories 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 an RDF repository and can be used to instantiate AM, DM, CM and data repositories. The properties of a repository (e.g. type of storage) are stored in an instance of the class DBInfo. SysConfig provides operations to parse a Hydragen configuration File.
- *nl.tue.heras.utils*: This package contains Hydragen-specific exception classes, and some utility classes that are used throughout the rest of the packages..

There are two distinct phases in running Hydragen. The initialization phase is initiated by HydragenWeb, whereas the execution phase involves the request-generate-respond cycle.

Upon initialization, HydragenWeb initializes HydraGenCore, which loads the various models from a local file or sets up connections to the given remote repositories (depending on the settings in the server configuration file). Subsequently, HydragenWeb loads the Presentation Model, which in this case is an XSLT file with some predefined variables.

The models loaded in Hydragen are checked both for syntactic and semantic validity. The syntax validity check is left to the Sesame RDF parser, which reports any errors in the models or data. For semantic checking we use the Pellet OWL-DL reasoner[18]. By using Pellet we are able to check whether the context data adheres to the Context Model, the content data adheres to the Domain Model and whether the Application Model adheres to the Application Meta-Model. Because of performance reasons, we use the validation capabilities of Pellet only in the initialization phase.

In the execution phase, the HydragenWeb servlet handles the http requests generated by the user. HydragenWeb implements both the GET and POST methods as required by the HTTP protocol, in order to support regular HTML requests and form submissions respectively. For each request, the generateAMP() method from the CoreEngine class is invoked with a reference to the requested Navigational Unit.

The generateAMP() method is responsible for processing the request for a specific Navigational Unit, 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, CM and data repositories before further processing the request. AMParser then initializes the AMP output file as an RDF file. Although a request is always for a specific Navigational Unit, the Application Meta-Model allows a Navigational Unit to contain Sub-Units and Set-Units; in order to distinguish between handling of the root unit and these sub-units, the handleRootUnit method is called. The root Navigational Unit is processed only once for a request, while Sub-Units can potentially be processed several times depending on what was specified in the AM.

6.3 Insert, Update and Delete

Since Hydragen supports the creation of new RDF nodes, deletion and updating of statements in the content/context data, the statements that are generated or modified during the processing of a Navigational Unit are captured in a set of internal lists: add-list, delete-list, and amp-list. As a last step before completing the AMP, these modifications are submitted. The additions and deletions (from add-list and delete-list) are submitted to the relevant RDF repositories, while the AMP statements (from amp-list) are first stored in the session-specific context of the AMP repository. After that, the contents of the session context of the AMP repository are written into RDF/XML format so that it can be fed to the XSLT processor.

Although there is support for SELECT and CONSTRUCT queries in SeRQL, it does not support update or delete queries (nor does SPARQL). Since updates and deletions are frequently required in the AM (e.g. to update context information, see example below), support for UPDATE and DELETE queries was added to Hydragen.

In order to facilitate the translation of these queries to SeRQL/Sesame queries and programming constructs, a specific syntax is used in UPDATE and DELETE clauses. The update expression must be wrapped inside an eval function, and the node that needs to be updated must be placed between "%" signs in the update expression. For example, consider the following query which updates the number of times a user has viewed a Movie page:

---

[18] http://clarkparsia.com/pellet

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

This query updates the number of views (+1) for user $U whenever he/she views the movie page for movie $M. An update query is processed as follows: the current value and type of the node are retrieved, the tuple with the current value of the node is deleted, the update expression is evaluated, and finally a new statement is created with the updated node.

7. Supporting SEAL in Hydragen

As HydraGen was initially not available, SEAL was first implemented as an independent module. Later, the SEAL implementation was integrated into HydraGen. In this section, we elaborate on both, and provide results of our initial experiments to measure performance overhead resulting from (runtime) SEAL aspects execution.

7.1 SEAL implementation

The parser for the pointcut part of the language was constructed using the JavaCC parser generator[19], while the Javacc JJTree tool was used to automatically generate AST (Abstract Syntax Tree) classes. This tool also provides support for the Visitor design pattern, which is used here to traverse the AST corresponding to a given pointcut expression. Sesame (Broekstra 2002) is used to store the AM and DM, and to execute SeRQL queries on them.

Our approach for pointcuts consists of translating each of the pointcut conditions (restrictions) to a SeRQL query, which extracts the elements from the AM that satisfy the condition. These queries are then executed and their results put in separate Vector objects. The logical connectors, combining the conditions, are mapped to equivalent Vector methods (e.g. logical conjunction corresponds to retainAll). The count function is implemented by executing the query corresponding to the nested condition, and subsequently counting the resulting values (note that there is no equivalent for 'count' in SeRQL). We took the approach of mapping conditions to separate queries, as opposed to translating the pointcut to one single SeRQL query, to avoid nested queries (which have known performance issues), and to uniformly implement references to the Domain Model.

The implementation of the advice uses a similar approach: the JavaCC parser generator and the JJTree tool were used to generate AST classes. The Visitor design pattern was also used to evaluate the advices, using the relevant SeRQL packages to alter the RDF(S)-based Application Model[20]. A separate package was implemented for SeRQL-query rewriting, implementing the strategy described in the previous section.

7.2 Integrating SEAL with Hydragen

Before the integration of SEAL and Hydragen could take place, several design decisions had to be made concerning the execution time of the aspects. As mentioned before, an aspect can use information from the Application Model, the Domain Model, Context Model and content/context data to implement

---

[19] Java Compiler Compiler [tm] (JavaCC [tm]) - The Java Parser Generator. https://javacc.dev.java.net/

[20] Note that our implementation here currently relies on SeRQL to modify RDF repositories. However, with update extensions to SPARQL becoming available, the latter would be equally usable.

the relevant adaptation requirements. Based on this observation, aspects may be divided into two categories: those can be executed at design time, and those that need runtime information before they can be executed. An aspect that only accesses the AM may be executed at design time. However, aspects that require information from the DM, CM or content/context data have to be executed at each page request, because this information is considered as volatile across requests. More specifically, aspects in the first category have pointcuts that only exploit AM information, and advices where either conditions are added to the queries of the selected elements, or where elements are unconditionally added, removed or replaced (see first two examples in 5.3). Aspects in the second category use DM (or CM) information in the pointcut and/or conditionally add, remove or replace elements in the advice (see third and fourth example in 5.3).

It can be argued that the DM (and the CM) can be considered much less prone to change than instance or context data, and therefore aspects that exploit such model information could also be executed at design time. However, this depends on the specific Web application: for example, in case of a Web 2.0-style application, new concepts can be added at runtime. In that case, the DM will be frequently altered, and aspects relying on DM information need to be executed at each page request. Therefore, the choice between design time or runtime execution of these types of aspects should be made configurable, so the choice can be delegated to the Web application developer.

As mentioned above, aspects in the first category can be executed before the HydragenCore engine is started. However, in that case an additional mechanism would be required to "undo" aspects. For example, each time an aspect needs to be undone, the new set of aspects (minus the aspect that is being undone) is executed on an original copy of the Application Model, which is then fed to the Hera-S engine to use as the new basis for generating AMPs. For ease of implementation, all aspects are currently executed at runtime upon page request. This implementation can later be optimized to include support for aspects in the first category.

One possibility for executing aspects per page request is by making a copy of the AM for each request, executing all aspects on this copy and subsequently selecting the requested Navigational Unit from the altered copy. However, this would require making a copy of the AM for *each* page request, which is prohibitively expensive. On the other hand, first extracting the requested Navigational Unit and subsequently executing the aspects on it is also not an option, because poincuts can use AM information not present in the unit (see first three examples in 5.3). Therefore, we have employed a strategy where the two parts of an aspect are executed on two different RDF repositories. Firstly, the pointcuts are executed on the entire Application Model, selecting the relevant elements. Then, the requested Navigational Unit is retrieved from the AM, and only the advices of aspects of which the selected elements are contained within the unit are executed.

Because the SEAL package and HydragenCore engine were developed concurrently at different locations, neither implementation had been specifically tailored for integration. In other words, the original implementation of HydragenCore did not provide an explicit hook for our implementation, nor was our implementation suited to work with the HydragenCore engine (even different versions of the Sesame repository were used). Most importantly, the engine directly retrieved the requested unit from the AM as a set of literal values, which were subsequently used to generate the AMP instance. However, for our purpose the requested unit first needs to be extracted as an RDF graph, so the relevant advices can be executed on it. Therefore, this part of the engine had to be altered in order to allow the runtime execution of aspects. In the current engine, the requested unit is first retrieved as an RDF graph; secondly, the aspects are executed using the aforementioned strategy; thirdly, the necessary values are retrieved from the unit to generate the AMP. The original SEAL implementation was altered to use the "handles" from the *nl.tue.heras.datatypes* package, which encapsulate the connection to the (possibly

remote) RDF repositories. Additionally, the execution of aspects had to be changed in SEAL, in order to support the aforementioned strategy of executing aspects per page request (i.e. the separate execution of pointcuts and advices).

7.3 SEAL execution performance

In order to gain some insight on the performance overhead caused by SEAL aspect execution, we set up a small-scale experiment. In this experiment, we deployed our HydragenWeb application and a local Sesame server (responsible for providing the data models and content) on a Dell Inspiron 6400 with an Intel Core 2 Duo T7200 processor and 2 Gigabytes of memory. Also, we created a Web application based on our IMDB running example mentioned in section 3 (i.e. 296.233 RDF triples, consisting of 23.626 movies and 1481 persons). We defined the following three aspects for our experiments:

1/ POINTCUT: hasType subUnit and hasName "*MovieSearchResultElementUnit";
ADVICE: ADD CONDITION imdb:hasUserRating.averageVote >= 6;

2/ POINTCUT: hasType subUnit and contains (count(hasType attribute) >= 3);
ADVICE: REPLACE subUnit BY relationship;

3/ POINTCUT: hasType subunit and hasInput in [SELECT I FROM {I} {imdb:hasAgeRating} {J}]
ADVICE: ADD CONDITION imdb:hasAgeRating.minAllowedAge <= cm:age;

Experiment 1:

As our aspect-execution is done on page request, we restricted our measurements to one single page: the MovieSearchResultUnit (a unit showing the results of a movie search). In Test 1, we measured the time required to generate AMPs[21] without aspects (Test 1.1) and with aspects (Test 1.2). Quite unexpectedly, the aspect version of the HydragenWeb application appears almost twice as fast in this measurement. However, this can be explained by our choice of aspects and unit: because aspect 1/ and 3/ make the search result queries more restrictive, the queries adapted by these aspects produce fewer results (which are retrieved from the sesame server), which leads to a faster AMP generation. In order to test this hypothesis, two additional tests were done: one that separately measured the execution time of the search result query (Test 2.1 and 2.2, denoting the execution without and with aspects, respectively), and one that measured the time needed to extract the requested Navigational Unit (see 7.2) and execute the aspects on it (Test 3). From these results, it is indeed clear that the more general (non-adapted) queries result in much longer execution times (Test 2.1 versus Test 2.2), and that this more than compensates for the overhead of executing aspects (Test 3).

Test 1.1	Test 1.2 (aspects)	Test 2.1	Test 2.2 (aspects)	Test 3
1055,784	508,036	656,928	64,410	167,711

Table1. Results of Experiment1 (averages in milliseconds over 10 runs)

Experiment 2:

Based on the observations from the previous experiment, we specifically altered the advice of the third aspect, so the aspect will no longer result in more restrictive search queries (only the third aspect was

[21] As aspects operate on AM level, they do not interfere with HTML generation. Hence we excluded this process from our measurements.

used in this experiment). The results can be seen in the table below (Test 4.1 and 4.2, denoting the execution without and with aspects, respectively): we now see a slight performance loss for the aspect version. We expect this performance loss to increase as the overhead of extracting the requested Navigational Unit increases (e.g. as larger units are requested), and more (and larger) aspects are employed.

Test 4.1	Test 4.2 (aspects)
903,356	989,180

Table2. Results of Experiment2 (averages in milliseconds over 10 runs)

8. Conclusion

In this article, we pursued a further separation of design concerns in WIS design, by separating the adaptation engineering process from the regular Web design process. In the realization of this aim, we made two key observations: (i) adaptation is typically spread throughout the Web application (design), resulting in a cross-cutting design concern, and (ii) semantic meta-data associated with the content is becoming increasingly available, allowing for powerful and flexible adaptation support. The first observation led to the choice of aspect-orientation to tackle adaptation engineering; the second observation inspired the use of semantic meta-data as a foundation for adaptation specification. The combination of these two key elements resulted in a semantics-based, aspect-oriented approach to adaptation engineering, materialized in the form of a domain-specific language called SEAL. We demonstrated in this article that this combined approach yields several advantages:

- *Separation of adaptation engineering*: by using the aspect-oriented paradigm, the selection of elements and the actions to be performed upon them can be encapsulated in so-called aspects, effectively separating the implementation of adaptation concerns from the actual application design. Moreover, this allows the different adaptation concerns to be separated from each other as well.
- *More powerful*: by allowing element selection via a query over the AM, the explicit enumeration of relevant elements can be avoided. Furthermore, by providing the possibility to reference properties from the underlying domain and global (structural) properties of the AM, elements can be selected in a powerful and expressive way.
- *More flexible and robust*: because pointcuts can select elements dynamically based on semantic meta-data (or global AM properties), the adaptation specifications are less vulnerable to errors when the domain gets extended or the application (design) changes. This is not the case for condition- or rule-based approaches, where the relevant elements for an adaptation concern have to be manually identified.
- *Less error-prone*: because the manual selection of relevant Web page elements upon which to perform adaptation is effectively avoided, human error can be significantly reduced. This is especially the case when the original application is extended/changed, which would otherwise require us to inspect all adaptation specifications, and extend/alter some of them to reflect the changes in the application.

- *More compact*: SEAL is a domain-specific language constructed with the specific structure of the Hera-S models in mind. As a consequence, it is more compact and allows for simpler adaptation specification.
- *Better re-usable*: pointcuts that select elements based on semantic properties and/or relations in the underlying domain are not wired to one particular application in any way, and can thus be re-used across different applications. Similarly, pointcuts that are based on global properties of application (model) are not specific to one application and facilitate re-use.

We presented our work in the context of an existing WIS design method, Hera-S. To demonstrate the viability of our approach, the implementation generation tool for Hera-S, named HydraGen, was extended with support for SEAL. The HydraGen implementation builds on existing Semantic Web technologies (i.e. RDF(S)/OWL) and tools (i.e. Sesame), of which the querying capabilities were exploited to realize the aspect-oriented adaptation specification for SEAL. Initial performance tests showed that aspect execution caused some overhead. However, when aspects introduce restriction (which they often do), a performance gain is obtained as the resulting restricted queries return fewer results, thus speeding up page creation.

Although the presented approach offers significant advantages compared to existing approaches, the introduction of aspect-orientation also raises an issue that currently has not been sufficiently investigated: the possible interaction(s) that can occur between different adaptation aspects. Indeed, theoretically it is possible that one adaptation aspect counteracts or even nullifies the effects of another. We have experimented with adding priorities to aspects in order to grant the adaptation engineer control over the execution order of aspects. Although viable, this solution was not completely satisfactory, as different adaptation concerns are ideally addressed independently of each other: therefore, the adaptation engineer has no clear view on possible interactions when describing a certain adaptation specification. Consequently, a more effective solution to avoid unwanted aspect interactions is required (e.g. by analyzing aspects of which the pointcut expressions lead to overlapping result sets), and is considered future work. Finally, the experiments presented in this article only hint on performance and scalability. More rigorous analysis and extended experiments, taking into account all variables (size of AM and DM, size of dataset, type of aspect, number of simultaneous users, etc), are required to clearly estimate scalability of the presented approach.

References

Bausmeister, H., Knapp, A., Koch, N., Zhang, G, 2005. Modelling Adaptivity with Aspects. In Proceedings ICWE2005, Sydney, Australia, 406-416.

Brambilla,M., Comai,S, Fraternali, P., Matera,M., October 2007. Designing Web Applications with WebML and WebRatio. In Web Engineering: Modelling and Implementing Web Applications (Human-Computer Interaction Series), Eds. G. Rossi, O. Pastor, D. Schwabe, L. Olsina, Springer, ISBN: 978-1846289224.

Broekstra, J., Kampman, A., van Harmelen, F, 2002. A generic architecture for storing and querying rdf and rdf schema. In Proceedings of ISWC 2002, LNCS 2342, 54–68.

Brusilovsky, P, 1996. Methods and techniques of adaptive hypermedia. In User Modeling and User-Adapted Interaction, 6 (2-3), Springer Science+Business Media B.V, 87-129.

Brusilovsky, P., 2001. Adaptive hypermedia. User Modeling and User-Adapted Interaction, 11(1-2), 87-110.

Casteleyn, S., De Troyer, O., Brockmans, S, 2003. Design Time Support for Adaptive Behaviour in Web Sites, In Proceedings of the 18th ACM Symposium on Applied Computing, , Melbourne, USA, 1222-1228.

Casteleyn, S. 2005. Designer-Specified Self Reorganizing Websites, Phd thesis, Vrije Universiteit Brussel.

Casteleyn, S., Fiala, Z., Houben, G.J., van der Sluijs, K, 2006. Considering Additional Adaptation Concerns in the Design of Web Applications, Adaptive Hypermedia and Adaptive Web-Based Systems, In Proceedings of the Fourth International Conference, AH2006, Dublin, Ireland, 254-258.

Casteleyn, S. Van Woensel, W., Houben, G.J, 2007. A Semantics-based Aspect-Oriented Approach to Adaptation in Web Engineering, In HT'07, Proceedings of the Eighteenth conference on Hypertext and hypermedia, Manchester, UK, 189-198

Ceri, S., Fraternali, P., Maurino, A., Paraboschi, S, 1999. One-To One Personalization of Data-Intensive Web Sites. In WebDB Workshop.

Ceri, S., F. Daniel, V. Demaldé, and F. M. Facca, 2005. An Approach to User-Behavior-Aware Web Applications. In Proceedings of the Fifth International Conference of Web Engineering, Sydney, Australia.

De Bra, P., Smits, D., Stash, N. The Design of AHA!, 2006. In Proceedings of the Seventeenth ACM Conference on Hypertext and Hypermedia, Odense, Denmark, 171-195.

De Troyer, O., Casteleyn, S, 2001 The Conference Review System with WSDM, First International Workshop on Web-Oriented Software Technology, IWWOST'01, (also http://www.dsic.upv.es/~west2001/iwwost01/), Eds. Oscar Pastor, Valencia (University of Technology), Spain .

De Troyer, O., Casteleyn, S., Plessers, P, 2007. WSDM: Web Semantics Design Method, Chapter 11 in Web Engineering: Modelling and Implementing Web Applications, Human-Computer Interaction Series Eds. Gustavo Rossi, Oscar Pastor, Daniel Schwabe, Louis Olsina, Springer, ISBN 978-1-84628-922-4, Vol. 12, 303-352.

Fiala, Z., Hinz, M., Meissner, K., Wehner, F, 2003. A Component-based Approach for Adaptive Dynamic Web Documents. Journal of Web Engineering, Vol. 2, No. 1&2, 58-73.

Fiala, Z., Frasincar, F., Hinz, M., Houben, G.J., Barna, P., Meissner, K, 2004. Engineering the Presentation Layer of Adaptable Web Information Systems, In Proceedings of the Fourth International Conference on Web Engineering, Munich, Germany, 459-472.

Garrigós I., Gómez J., Barna P., Houben G.J, 2005. A Reusable Personalization Model in Web Application Design. In Proceedings of the International Workshop on Web Information Systems Modeling (WISM), Sydney, Australia.

Garrigós I., Gómez J, June 6, 2006. Modeling User Behaviour Aware Websites with PRML. In Proceedings of the International Workshop on Web Information Systems Modeling (WISM 2006), Luxembourg

Houben, G.J., Van der Sluijs, K., Barna,P., Broekstra, J., Casteleyn, S., Fiala, Z., Frasincar, F, 2007. Hera, Chapter 10 in Web Engineering: Modelling and Implementing Web Applications, Human-Computer Interaction Series, Eds. Gustavo Rossi, Oscar Pastor, Daniel Schwabe, Louis Olsina, Springer, ISBN 978-1-84628-922-4, Vol. 12, 263-301.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, J., Irwin, J, 1997. Aspect-Oriented Programming. In Proceedings of the 11th European Conference on Object Oriented Programming (ECOOP'97), Jyväskylä, Finland, 220-242.

Koch, N., Kraus, A. and Hennicker, R, 2001. The Authoring Process of the UML-based Web Engineering Approach. In Proceedings of the 1st International Workshop on Web-Oriented Software Technology.

Kraus,A., Knapp,A., Koch, N., 2007. Model-Driven Generation of Web Applications in UWE. In 3rd International Workshop on Model-Driven Web Engineering (MDWE 2007), CEUR Workshop Proceedings, Vol. 261.

Pastor,O.,Fons, J., Pelechano V., and Abrahão, S., 2006. Conceptual Modelling of Web Applications: The OOWS Approach", In Web Engineering, editors Emilia Mendes and Nile Mosley, Springer Berlin Heidelberg, ISBN 978-3-540-28196-2, 277-302.

Rossi, G., Schwabe, D., 2007. Modelling and Implementation Web Applications with OOHDM, Chapter 6 in Web Engineering: Modeling and Implementing Web Applications, Human-Computer Interaction Series Eds. Gustavo Rossi, Oscar Pastor, Daniel Schwabe, Louis Olsina, Springer, ISBN 978-1-84628-922-4, Vol. 12, 109-156.

Schauerhuber A., Wimmer M., Schwinger W., Kapsammer E., and Retschitzegger W., March 2007. Aspect-Oriented Modeling of Ubiquitous Web Applications: The aspectWebML Approach. In Proceedings of the 5th Workshop on Model-Based Development for ComputerBased Systems: Domain-Specific Approaches to Model-Based Development, in conjunction with ECBS'07, Tucson, AZ, USA.

Schwabe, D. and Rossi, G, 1998. An object oriented approach to web-based applications design. In Theory and Practice of Object Systems, John Wiley & Sons, Inc , ISSN 1074-3227, 4(4), 207-225.

Schwabe, D., Szundy, G., De Moura, S.S., Lima, F, 2004. Design and Implementation of Semantic Web Applications. In WWW Workshop on Application Design, Development and Implementation Issues in the Semantic Web

Schwinger, W., Retschitzegger, W., Schauerhuber, A., Kappel, G., Wimmer, M., Proll, B., Cachero, C., Casteleyn, S., De Troyer, O., Fraternali, P., Garrigós, I., Garzotto, F., Ginige, A., Houben, G.J., Koch, N., Moreno, N., Pastor, O., Paolini, P., Ferragud, V.P., Rossi, G., Schwabe, D., Tisi, M., Vallecillo, A., van der Sluijs, K., Zhang, G, 2008. "A survey on web modeling approaches for ubiquitous web applications", In International Journal of Web Information Systems, , Eds. David Taniar and Ismail Khalil Ibrahim, Publ. Emerald, ISBN 1744-0084, Vol. 4, No. 3, 234-305.

Van Deursen, A., Klint, P. and Visser, J, 2000. Domain-Specific Languages: An Annotated Bibliography. In SIGPLAN Notices 35(6), ACM Press, 26-36.

Van der Sluijs, K., Houben, G.J., Broekstra, J., Casteleyn, S. Hera-S, 2006. Web Design Using Sesame, In Proceedings of the Sixth International Conference on Web Engineering, Palo Alto,California, USA, 337-344.