A Semantics-based Aspect-Oriented Approach to Adaptation in Web Engineering

Sven Casteleyn
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussel, Belgium
Sven.Casteleyn@vub.ac.be

William Van Woensel
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussel, Belgium
William.Van.Woensel@vub.ac.be

Geert-Jan Houben
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussel, Belgium;
Technische Universiteit Eindhoven,
PO Box 513, 5600 MB Eindhoven,
The Netherlands
Geert-Jan.Houben@vub.ac.be

ABSTRACT
In the modern Web, users are accessing their favourite Web applications from any place, at any time and with any device. In this setting, they expect the application to user-tailor and personalize content access upon their particular needs. Exhibiting some kind of user- and context-dependency is thus crucial in Web Engineering. In this research, we focus on separating the adaptation engineering process from regular Web engineering by applying aspect-oriented techniques. We show how semantic information and metadata associated with the content can be exploited in our aspect-oriented approach. Furthermore, the approach allows the use of global (structural) properties of the Web application in adaptation specification. We thus obtain several advantages, which are demonstrated in this paper: to control adaptation (specification) separate from (regular) Web Engineering concerns in a richer, more consistent, robust and flexible way.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architectures – Languages; H.5.4 **[Information Interfaces and Presentation]**: Hypertext/Hypermedia – Architectures, Navigation.

General Terms: Design, Languages

Keywords
Web Engineering, Aspect-Orientation, Adaptation, Semantic Web.

1. INTRODUCTION
Technology developments and evolutions on the Web bring new requirements and challenges for Web engineers, as well as opportunities.

Wireless network technology has been improving rapidly over recent years, and finally seems 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. This trend in technology is one of a number of examples that motivate the need for advanced user and context adaptation in Web applications. Moreover, the trend and its amazing speed also clearly indicate the need to control the adaptation engineering within Web application engineering.

A parallel evolution is the emergence of the Semantic Web, in which the semantics of the content are made explicit by means of metadata and ontologies. With more and more semantic data available (mostly in the form of RDF(S), also in OWL), the steps toward realizing the Semantic Web are becoming more and more concrete. Just to point out some examples: Wordnet has been available in RDF format some time now[1]; Wikipedia can be accessed in various RDF-notations[2], and evidently modern Web 2.0 applications (e.g. YouTube, MySpace, Flickr) provide a wealth in user-provided meta-information in the form of tagging. This wealth of semantic (metadata) information leads to exciting new possibilities: semantic-based search engines[3], automated agents crawling and extracting relevant information, complex querying[4], (automatically) combining information from different sources[4], etc. These examples exploit Semantic Web techniques and approaches that offer opportunities for Web engineers as well. One opportunity that deserves more attention lies in exploiting this metadata to perform more flexible and complex

HT'07, September 10–12, 2007, Manchester, England, United Kingdom.

[1] See http://www.w3.org/2001/sw/BestPractices/WNET/wn-conversion.html

[2] See http://labs.systemone.at/wikipedia3

[3] E.g. http://www.swoogle.com/

[4] E.g. http://dbpedia.org/

personalization and adaptation in access to the application content[5].

The aforementioned heterogeneity of the Web and client devices, and the growing demand and expectation of the user for a personalized browsing experience that depends on location, device and time, naturally have their consequences for the Web engineer. To provide content access that thus adapts to user and context the Web engineering approaches require straight-forward but rigid, powerful and independent adaptation engineering. Next, we observe that adaptation is typically cross-cutting: instead of being localized to one particular place in the Web application (design), it is typically spread over the entire Web application. In regular software engineering, aspect-orientation is a proven abstraction and modularization technique to tackle so-called cross-cutting design concerns. Consequently, it is our proposition that these techniques can serve excellently to tackle (cross-cutting) adaptation concerns, and separate the adaptation specification from the regular Web engineering (concerns). At the same time, aspect-oriented adaptation reduces the required effort when specifying adaptation, mainly by preventing unnecessary (adaptation) code duplication. In this paper, we concretely materialize the aspect-oriented techniques by exploiting the semantic metadata available in (the design of) the Web applications content. By doing so, we successfully separate adaptation specification from regular web design, and obtain several advantages, as will be argued in this paper: simpler yet more flexible, more robust, consistent and powerful adaptation support, and a higher degree of re-usability.

This paper is structured as follows. Section 2 shortly reviews Hera-S, the Web engineering method used to illustrate our approach. Section 3 shortly elaborates on existing adaptation support in Hera-S. Section 4 explains why and how aspect-orientation is used to specify complex adaptation, based on semantic (meta)data. Section 5 discusses how to realize the approach presented in this paper. Section 6 discusses our implementation, and finally section 7 states conclusions.

2. Hera-S in a Nutshell

Hera-S is a Web Information System (WIS) design method that combines the strength of Sesame [3], a popular open source RDF framework, and the rich modeling capabilities of Hera, a model-driven approach for engineering Web applications based on semantically structured data. As most existing WIS design methods, Hera-S distinguishes domain, navigation and presentation design. By cleanly 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: it takes as input an RDF- or OWL-based specification of the content data (the so called *Domain Model*), and specifies how the data is chunked and made navigable in the so-called *Application Model*. The *Presentation Model* finally adds the necessary details for presentation (e.g. positioning, look-and-feel). We remark that the schemas of these models are also specified using RDF(S). Another distinctive characteristic of Hera-S is its explicit focus on adaptation and personalization. Hera-S therefore maintains a *Context Model*, which allows user- and context-based adaptation (more on this in the next sections). 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 and is stored as a Sesame repository, the Application Model is instantiated, resulting in so-called *Application Model Pages* (AMP's). Using the presentation model, the actual Web pages can be generated. In previous publications, it was demonstrated that both proprietary and external engines can be used for this task (e.g. [11]).

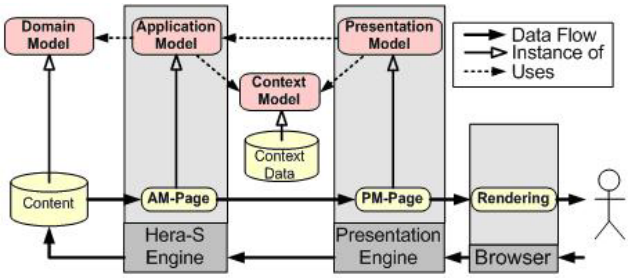

Figure 1 Hera-S Architecture

To later understand the semantic-based adaptation illustrations expressed in Hera-S, let us consider some small examples, taken from our running case, namely the IMDB Website[6]. 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 paper. To give a small idea of it's scale: the Domain Model (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 Application Model (AM), where adaptation will be specified. When exploiting the semantics of the content data to perform adaptation, we will also use the Domain Model (DM), but as this is simply standard RDF(S) it does not require any further clarification. Our examples are expressed using Turtle[7], 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 will be shown together to the user. Units typically correspond to a single concept in the DM, yet this is not mandatory (i.e. they might also refer to several domain concepts). The elements contained in a Unit are either Attributes, or other Units (called Sub-units). Attributes are literal values, typically representing strings although any other URI-referable media type is allowed (e.g. pictures, videos). Attributes mostly refer to (datatype) properties in the DM[8], and thus specify exactly which of these properties will be presented to the user. Sub-units

[5] See for example http://www.personal-reader.de/

[6] http://www.imdb.com/

[7] http://www.dajobe.org/2004/01/turtle/

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

simply provide further grouping of elements, and behave much like regular Units. Finally, Relationships represent (browsable) links between units, and are built upon underlying semantic relationships between concepts of the DM. 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 its director(s) (his/her name, picture and movies he/she directed). In Turtle notation, the AM includes:

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

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

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

  am:hasUnit [
    rdfs:label "Director" ;
    am:refersTo :DirectorUnit ;
    am:hasQuery
     "SELECT D FROM {$M} rdf:type {imdb:Movie};
      imdb:movieDirector {D} rdf:type {imdb:Director}"
] .

:DirectorUnit a am:NavigationalUnit ;
  am:hasInput [ a am:Variable ;
                am:varName "D" ;
                am:varType imdb: Director];

  am:hasAttribute [
    rdfs:label "Name" ;
    am:hasQuery
      "SELECT L FROM {$D} rdf:type {imdb:Director};
                         rdfs:label {L}" ] ;

  am:hasAttribute [
    rdfs:label "Photo" ;
    am:hasQuery
      "SELECT Ph FROM {$D} rdf:type {imdb:Director};
                          imdb:hasMainPhoto {Ph}" ] ;

  am:hasRelationship [
    rdfs:label "Movies directed" ;
    am:refersTo :MovieUnit ;
    am:hasQuery
      "SELECT M FROM {$D} rdf:type {imdb:Director};
                          imdb:directorFilmography {M}"
] .
```

The example consists of two units: MovieUnit and DirectorUnit, both containing some elements and representing a particular grouping of information. The MovieUnit contains two attributes, one representing the "title" of a movie, the other representing the "plot" of the movie. Finally, the MovieUnit also contains a sub-unit, representing the director(s) for that movie. Note that the sub-unit "refersTo" the DirectorUnit, which specifies what exactly to show for a Director. We chose here to use a sub-unit, as opposed to a navigational relationship, which will result in the final Webpage showing the information on directors shown on the same page as the movie (instead of a link to a page representing the director). The DirectorUnit also contains some attributes, and a relationship, which represents a browsable element, in this case linking to the movies directed by this director.

Another issue to remark for this example is that (almost) each element in a unit, be it a sub-unit, an attribute or a relationship, has a SeRQL query associated with it. This query is responsible for the instantiation of the particular AM-element with actual data (from the data source), and allows the designer to select exactly and only the particular content that is desired. Thereby, the designer wields the complete power of SeRQL, and is also able to integrate Context Model data in the query. In this way, as we will see in the next section, versatile and fine-grained adaptation can naturally be integrated in the Application Model.

Next to the above basic constructs, Hera-S supports update-queries (to update the context model), forms, scripts, (guided) tours, Web services and frame-based navigation. For details on these elements we refer to [16].

3. Adaptation Engineering Perspective

We now take a closer look at the adaptation specification. Before we continue with our Hera-S running example, let's shortly review how adaptation is typically tackled in hypermedia in general, and in some well-known Web engineering methods in particular. Adaptation specification in hypermedia lead to the field of *Adaptive Hypermedia*, where both techniques and methods for adaptation were identified [4]. In this context, adaptive access to content is mostly achieved by adding (adaptation) conditions to the regular hypermedia design. These conditions constitute adaptive access to hypermedia content: depending on the truth-value of these conditions, some content is either shown or hidden (a representative example is the AHA! System [2]). 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 [15]. In WebML, navigational conditions are specified as queries over de data [7] in a so called profile. Some methods apply Event-Condition-Action rules to specify adaptation. Examples include OO-H, which uses the Personalization Rule Modeling Language (PRML) [12], WSDM, which uses the Adaptation Specification Language [5] and WebML, in the context of user-behavior aware Web Applications [8]. It is worth noting that some of these methods do consider other actions besides conditioning elements to be viewed, e.g. sorting, altering navigation paths. UWE, an OO-based Web engineering method, specifies adaptation conditions as OCL constraints on the conceptual model [14]. We do remark that, in the context of UWE, aspect-orientation has been used to specify (some) adaptation [1]. In that paper, (only) some particular types

of adaptive navigation were considered, i.e. adaptive link hiding, adaptive link annotation, adaptive link generation. Moreover, the granularity of the considered aspects is totally different: where [1] considers one particular kind of adaptation (e.g. link hiding) as an aspect, our paper considers the adaptation process as a whole. Aspect specification in our paper is thus considered at a higher level of abstraction, and exploits semantic information as a key element. Furthermore, an important drawback of [1] 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.

To illustrate how adaptation is specified in Hera-S, we use the same AM examples as introduced in the previous section. Flexible and expressive adaptation support is provided there through the SeRQL expressions underlying every element of the Application Model. Indeed, these expressions may include conditions that make 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. Reconsider the DirectorUnit from Section 2, and 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 am:NavigationUnit ;
  ...
  am:hasAttribute [
    rdfs:label "Photo" ;
    am:hasQuery
      "SELECT Ph
       FROM {$D} rdf:type {imdb:Director};
                 imdb:hasMainPhoto {Ph},
                {} rdf:type  {cm:CurrentUser};
                cm:userDevice {Dev}
       WHERE NOT Dev LIKE 'pda'"
   ] ;

  am:hasRelationship [
    rdfs:label "Movies directed" ;
    am:refersTo :MovieUnit ;
    am: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. 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 (manually) added also for these new parts of the AM.

- *Global (structural) properties are not usable:* because adaptation conditions are limited to one particular element (i.e. integrated in a query), they cannot express restrictions based on global properties of the Application or Domain 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.

We also observe that valuable semantic information is ignored in a scenario as above. Although the semantic (type) information for media elements *is* present in the Domain Model, this knowledge is ignored. Instead, it is left to the designer to interpret which queries involve pictures, and typically he/she must perform all required adaptation manually. This leaves room for human error, with possible inconsistencies in the adaptation behavior as a result. Adaptation engineering is also totally intertwined with the (classical) Web engineering (concerns), complicating the overall design.

Adaptation engineering obviously requires a more systematic and higher-level approach. We also notice how the semantic information (naturally present and associated with the content) is there to exploit for this purpose. In the next section, we present an aspect-oriented and semantic-based approach to adaptation specification. Our approach effectively separates adaptation engineering from (regular) Web engineering. Furthermore, we also tackle the fact that adaptation concerns are typically spread over the entire Web application and exploit the valuable semantic information typically present in the content specification. Finally, the approach is also perfectly suited to take into account (global) structural properties of the global Web application.

4. 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 [13]: 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, 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* [9]. 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 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 explain the SEAL language constructs and their semantics; how the semantics of the adaptation aspects are realized (in Hera-S) is the subject of the next section.

4.1 Pointcuts

Pointcut expressions help to 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 following constructs are available: "unit", "subunit", "attribute", "relationship", "query", "form" "label", "query", "tour", "target", "source". Typical examples include:

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

Conditions: Next to type restriction, element selection in the pointcut may be restricted according to name, value (a property of an AM-element should have a certain value; see the third example where it is specified that the label of an attribute should have 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*). Where appropriate, string pattern-matching is allowed when conditioning values (see the first example below). Specifically for relationships, restrictions on source and target may be specified (using *from* and *to*). Logical expressions may be negated or combined using logical conjunction or disjunction; parentheses are used to alter default logical operator precedence. Typical examples include:

- 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")

Native calls: SEAL supports calls to the native underlying query language, in our case SeRQL, using the <SeRQL> construct. This hook is provided so that the expert user can still specify intricate queries which would otherwise not be possible with SEAL. It also allows querying and referencing the Domain Model, and thus to exploit valuable DM information (we demonstrate this benefit in section 4.3).

4.2 Advices

Advices specify exactly what needs to be done to the element(s) selected in the pointcut. SEAL supports the following constructs:

Adding conditions: Conditioning elements is the common way of adapting. Conditions (condition expressions) typically reference the Context Model to express a kind of context or user-dependency. In analogy with Hera-S notational conventions (see section 3), referencing the Context Model is done using the "cm:"-prefix; referencing the Domain Model is done using the namespace-prefix that was used in the Domain Model (i.e. in our case "imdb:"). Navigation in the respective models can be done using the "."-operator (e.g. imdb:hasAgeRating.minAllowedAge; see 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, a property specified in the Domain Model. Note that in the left part, navigation starts from the elements selected in the pointcut, while

the right part specifies navigation in the Context Model (denoted by the "cm:" prefix) for the current user. We will elaborate further on this example in section 4.3.

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 in 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 hasNavigationRelation; all attributes of the particular sub-units are left unchanged)

4.3 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 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 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 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 certain 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 to a registration page

Adaptation Aspect:
POINTCUT type relationship and from (type subunit) and to (type unit)
ADVICE if cm:isRegistered = false REPLACE hasTarget value "*" BY hasTarget value "imdb: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 "imdb:RegistrationUnit" unit, causing all selected links to redirect 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 metadata 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
(<SeRQL> SELECT I FROM {imdb:hasAgeRating} rdfs:domain {I})
ADVICE ADD CONDITION
imdb:hasAgeRating.minAllowedAge <= cm:age;

The pointcut selects all sub-units[9] that have as input a certain concept which has a 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 Model) is higher than the minimum allowed age (specified in the Domain Model) 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, it will be determined which elements 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 metadata 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. In our IMDB example, 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:

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, if the user is using a pda.

Adaptation Aspect:
POINTCUT type subunit and contains (count (type attribute) >= 5) ;
ADVICE if cm:userDevice = "pda"
REPLACE hasSubunit BY hasRelationship

The pointcut selects all sub-units which have five or more attributes ("large" sub-units). The advice subsequently replaces these sub-units (actually their in-page occurrence) by relationships that link to dedicated pages representing this particular information. 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. 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 adding conditions to (single) elements. 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.

5. REALIZING ASPECT-ORIENTED ADAPTATION SPECIFICATIONS

As we have demonstrated in the previous section, specifying (metadata-based) aspect-oriented adaptation yields several advantages. However, as is always the case when offering more powerful and higher-level tools to the designer, the implementation strategy of these tools becomes more complex. Let us sum up the specific challenges, originating from our semantics-based aspect-oriented approach and discuss our considerations for implementing. In the next section, we will discuss our actual implementation.

The main challenge originating from our aspect-oriented approach is the fact that we allow the designer to specify, with one adaptation aspect, adaptation occurring at multiple places of the Application Model. Just as in aspect-oriented software development, this requires *weaving* the aspects in the regular code. In our case, two approaches are possible: either performing the weaving on model-level, and subsequently using the modified models with the regular Hera-S presentation generation engine (as shown in Figure 1), or realizing the adaptation aspects on instance level, in other words on the AM (instance) Pages resulting from the Hera-S presentation generation engine. The latter approach is feasible, and has already been implemented using a third-party rule-based adaptation engine (the GAC [10]) in the past, yet for a simpler version of SEAL. This solution consisted of mapping the adaptation aspects to rules, which were run on the AM Pages and performed the required adaptation [6]. However, this approach suffers from two major drawbacks: 1) too much data is retrieved: data that will be filtered out anyway by the adaptation rule(s) is still retrieved at every page request, yielding extra unnecessary and costly (real-time) processing, and 2) only filtering is supported: adding or replacing elements was impossible, unless the adaptation engine is connected to the Hera-S engine and a feedback loop is realized, which would again lead to loss of

[9] 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, a expression in the pointcut to also select these relationships (similar to the one selecting sub-units).

performance. The first approach, weaving the adaptation on model-level, does not yield these disadvantages, yet requires implementing an interpreter for SEAL-pointcuts, and injecting the adaptation conditions (specified in the advices) into arbitrary existing SeRQL queries corresponding to the elements identified in the pointcut (i.e. query rewriting). The former is done by evaluating the pointcuts, and translating them to SeRQL queries on the (RDF-based) Application Model, selecting the desired AM elements. The latter is done by extending the existing SeRQL queries (corresponding to each element of the AM selected in the pointcut), so that the existing path expressions (in the FROM-clause of the SeRQL-query) are extended to include the necessary elements on which we can specify the conditions. The actual conditions can then be written in (an additional) WHERE-clause. Since the path to the relevant property values and their comparison is specified in the advice condition, we have all the ingredients available to perform this query extension.

Another distinctive feature of our adaptation engineering approach is its metadata-based nature. Since knowledge contained in the Domain Model can be used by the adaptation engineer in the adaptation aspects, evaluating such adaptation aspects requires resolving these domain-references. Furthermore, adaptation advices are not limited to injecting conditions (e.g. adding/deleting/replacing elements), and may also contain Context Model references. The result is that such adaptation aspects cannot be applied at the time of deployment, but need to be applied at runtime, upon page-request. Fortunately, in this case the advice only needs to be applied on the requested page, not on the entire model, and thus performance overhead is minimal. The resulting (altered) part of the AM specification is then fed to the Hera-S presentation engine, which subsequently generates the requested (and adapted) Webpage.

6. IMPLEMENTATION

The parser for the pointcut part of the language was constructed using the JavaCC parser generator[10], 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 [3] 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.

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

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 used to evaluate the advices, using the relevant SeRQL packages to alter the RDF(S)-based Application Model[11]. A separate package was implemented for SeRQL-query rewriting, implementing the strategy described in the previous section.

The existing Hera-S presentation generation engine subsequently is to be used for actual presentation generation. Description of this tool is outside the scope of this paper. Adaptation aspect evaluation as described above is integrated in the tool, to be performed upon page request, before feeding the resulting (adapted) AM Page specification to the engine.

7. CONCLUSION

In this paper, a further separation of concerns in WIS design was pursued, by separating the adaptation engineering process from the regular Web design process. Based on a first observation that adaptation is typically a cross-cutting design concern, and the consequent observation that semantic metadata and its querying and access facilities are more readily available today, we proposed a semantics-based aspect-oriented approach to separate adaptation specification from the regular Web design. We demonstrated in this paper that this combined approach to adaptation engineering allows easier, more compact and more powerful adaptation specification for cross-cutting adaptation concerns. We showed that adaptation on the basis of semantic (meta)data results in more generic, more robust, more powerful and better re-usable adaptation specifications. We discussed in this paper our considerations related to the required implementation strategy, and finally our implementation, based on existing Semantic Web tools (i.e. Sesame).

Although the presented approach yields all the aforementioned advantages, some future work is still required. Our immediate plans are both theoretical and practical. An issue that is currently not sufficiently investigated is the possible interaction(s) that can occur between different adaptation aspects. Indeed, theoretically it is possible that one adaptation aspect nullifies the effects of another. As an immediate solution, we are considering to add priorities to aspects, so that the adaptation engineer has control over the execution order of aspects. However, a more profound study is needed to examine all possible aspect interactions. Next, we are also looking into the implementation to further optimize the (SeRQL) query construction resulting from aspects. Since the aspects are executed at runtime, performance is critical. Finally, further experiments with our IMDB Website will possibly lead to extending SEAL for further expressiveness.

8. REFERENCES

[1] Bausmeister, H., Knapp, A., Koch, N., Zhang, G. Modelling Adaptivity with Aspects. In Proceedings ICWE2005, Sydney, Australia, pp. 406-416 (2005)

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

[2] De Bra, P., Smits, D., Stash, N. The Design of AHA!, Proceedings of the 17th ACM Conference on Hypertext and Hypermedia, pp. 171-195, Odense, Denmark (2006)

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

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

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

[6] Casteleyn, S., Fiala, Z., Houben, G.J., van der Sluijs, K. Considering Additional Adaptation Concerns in the Design of Web Applications, Adaptive Hypermedia and Adaptive Web-Based Systems, 4th International Conference, AH2006, pp. 254-258, Dublin, Ireland (2006)

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

[8] S. Ceri, F. Daniel, V. Demaldé, and F. M. Facca. An Approach to User-Behavior-Aware Web Applications. in Proceedings of ICWE 2005 , Sydney, Australia, (2005)

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

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

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

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

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

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

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

[16] van der Sluijs, K., Houben, G.J., Broekstra, J., Casteleyn, S. Hera-S - Web Design Using Sesame, In Proceedings of the 6th International Conference on Web Engineering, pp. 337-344, Palo Alto,California, USA (2006)