Harnessing the Power of Semantics-Based, Aspect-Oriented Adaptation for AMACONT

Matthias Niederhausen[1], Kees van der Sluijs[2], Jan Hidders[3], Erwin Leonardi[3], Geert-Jan Houben[2,3], and Klaus Meißner[1]

[1] Technische Universität Dresden, Chair of Multimedia Technology, 01062 Dresden, Germany
{matthias.niederhausen,kmeiss}@tu-dresden.de
[2] Eindhoven University of Technology
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands
k.a.m.sluijs@tue.nl
[3] Delft University of Technology
P.O. Box 5031, NL-2600 GA Delft, The Netherlands
{a.j.h.hidders,e.leonardi,g.j.p.m.houben}@tudelft.nl

Abstract. Adaptivity in web applications raises several concerns. One demands it to be decoupled from the actual application and at the same time wants to use very domain-specific terms for dividing the audience into groups. Two current trends, aspect-oriented programming and semantic web technologies, fit these requirements like a glove. In this paper, we present the AMACONT web modeling framework and use it as an example of how to extend such a framework to make use of these powerful technologies. The underlying concepts, however, can be applied to the modeling of adaptivity in general.

Keywords: adaptation, aspect-oriented programming, semantic data.

1 Introduction

The concerns that must be considered when developing a web application are numerous and often change over time, by current trends on the Web or the availability of new technologies. Two such concerns that have emerged in the last years are the demand for adaptivity to the context (e.g., serving content that has been prepared for a special audience), and the need to actually model this context exploiting rich, semantics-based models. While an application designer typically addresses the audience in a high-level and often domain-specific way (e.g., "premium customers"), the existing frameworks do rarely offer facilities for easily improving and extending this context model. Further, in most cases the web application as such does already exist and its application and adaptation logic should be extended with these additional concerns. This raises the question of how an existing web framework can be extended to provide accordant support.

In this paper, we present AMACONT, a web modeling framework that has its roots in a component-based document format. We show how we extended the existing

M. Gaedke, M. Grossniklaus, and O. Díaz (Eds.): ICWE 2009, LNCS 5648, pp. 106 – 120, 2009.

AMACONT framework with two independent mechanisms – aspect-oriented adaptation modeling and semantics-based adaptation – for addressing the concerns of adaptivity and context modeling. We believe that aspect orientation is a useful technique to model aspects of an application that stem from separate concerns and appear throughout the whole application, such as privacy and security. We further argue that semantics-based adaptation can help to bridge granularity differences, e.g., between some available, detailed individual customer information and the "premium customer" classification – the rules for this classification can be more flexibly encoded in an ontology. Also it allows for easier and more flexible inclusion of additional knowledge from external ontologies such as customer preferences or product information, a type of flexibility still lacking in most existing web engineering methods. The approach that we take for extending the previous component-based document transformer gives a general recipe for extending web application development tools with aspect-orientation and semantic context modeling.

2 A Document Format for Adaptive Web Applications

AMACONT aims at enabling web engineers to design web applications tailored to users' varying needs, device capabilities and context data such as the users' location. As such, it is comparable to other web application design methods like SHDM, WebML, WSDM, UWE, OO-H, OOWS and Hera-S (for a survey on these web modeling approaches, cf. [11]). These methods typically separate concerns in the design process by distinguishing several distinct, but related models. A typical distinction is made into the following phases: (1) the underlying data model of the data that is to be presented, (2) the application model that describes the web application logic and (3) the presentation model that describes the exact lay-out and presentation of the web pages. Moreover, they all either allow the (semi-)automatic generation of code from their models or can interpret and execute them directly.

AMACONT provides a document format and a runtime environment, allowing for both static and dynamic adaptations of content, layout and structure. Furthermore, mechanisms for modeling dynamically changing context information like users' device capabilities, preferences and locations are provided in order to guarantee that adaptation is based on up-to-date context models.

2.1 Document Format

Targeting reuse of content, AMACONT employs a component-based document format to model adaptive web applications [5]. As Figure 1 shows, components in AMACONT can be classified into three different layers, with an orthogonal fourth Hyperlink layer.

The most basic components stem from the bottom layer of *Media Components*. Examples of such elements are text fragments, images, CSS stylesheets and videos.

Often there are fragments that are closely related and typically used in conjunction, like an image with a textual caption. By defining such semantic groups on the *Content Unit* layer of AMACONT, authors can structure their applications.

Finally, there is the *Document Component* layer. It allows to structure a web application further by grouping Content Units into larger components and further into

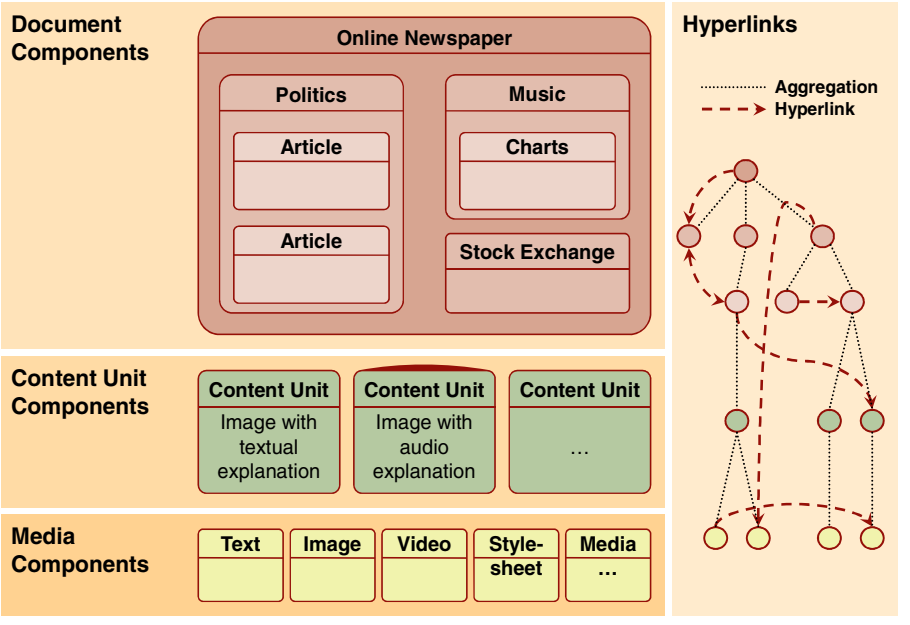

Fig. 1. The AMACONT Component Model

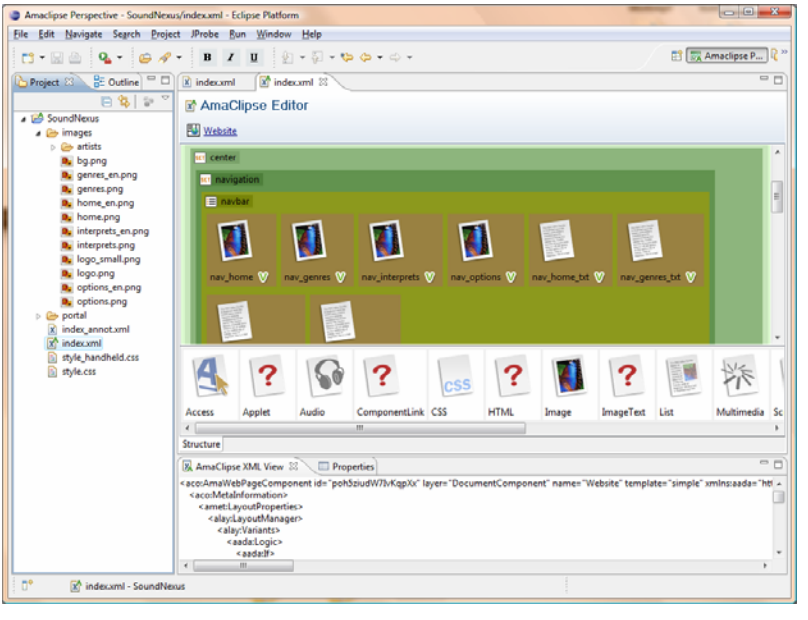

Fig. 2. The AmaArchitect Authoring Tool

web pages. AMACONT allows arbitrarily complex subcomponents, so that authors have the freedom to organize their page structure in any way they want. Note that web pages in AMACONT can be either modeled by structuring one document accordingly or by splitting the definition up into a number of documents, similar to HTML files.

Orthogonal to these three layers, the *Hyperlink* layer is spanned. Hyperlinks in AMACONT can reference components from all three levels. Thereby, it is easy to create a link starting from a composed page element. Links can target either an AMACONT file or an arbitrary component – or any custom URL.

In order to allow authors to easily create AMACONT applications, an Eclipse-based authoring tool, the AmaArchitect[1] has been designed and is constantly being improved. Figure 2 shows a screenshot of the tool, where in the middle, depending on the currently edited component, a specialized editor is shown. In the picture, this is an editor visualizing the document structure of the project's index page. At the bottom, we also have a quick XML overview of the currently selected component.

The next section explains how adaptivity can be defined on top of such a base document model in AMACONT.

2.2 Basic Adaptivity

As many other prominent web engineering approaches, AMACONT allows authors to specify adaptation by the means of alternative document fragments, called *variants*.

Variants can be defined for any components (or fragments thereof) on the four component layers. To this end, an adaptation construct must be created, consisting of two general parts: a *variants container* and a *selection condition header*. As its name suggests, the variants container contains alternatives for a given fragment, whereas the number of alternatives is not limited. Such a variant can also be empty, leading to the fragment's removal. The selection condition header specifies the contextual circumstances for showing each variant. At runtime, these conditions are evaluated to determine the variant to select. The details of this process are discussed in Section 2.3.

By using variants, authors can specify static adaptation, i.e., chose from variants that are known at design time. However, it is often desirable to include data that is only available at runtime or that changes over time, e.g., database contents. In this case, the actual content must be queried anew every time someone sends a request. To this end, AMACONT has been extended with a dynamic content engine that builds on component templates. Component templates are structured just like regular AMACONT components, allowing for a seamless integration of dynamic content. The difference, however, is that in their body they can refer to previously defined data fields. These data fields are typically defined in the component's head, by the means of an SQL query. It is possible to access the data fields of queries of all parent elements, which allows for an arbitrary nesting of queries. Further, component templates can be iterated: multiple copies of such a component are instantiated, each filled with different concrete values. By combining these features, it is possible to create dynamic linked lists (e.g., a list of all book authors together with all their books).

The next section explains the pipelined process of adapting and serving an AMACONT document in response to an HTTP request.

[1] The tool can be downloaded at `http://www.hyperadapt.net/AmaArchitect/`

2.3 Publication Process

For each user request, the respective AMACONT document which contains variants for all defined adaptation is fetched from the component repository and transformed in various steps, according to the current context. To the overall process there are three central elements: *document generation*, *context modeling* and *context acquisition*. Figure 3 shows an overview of the whole process, while the following subsections deal with the details of each component.

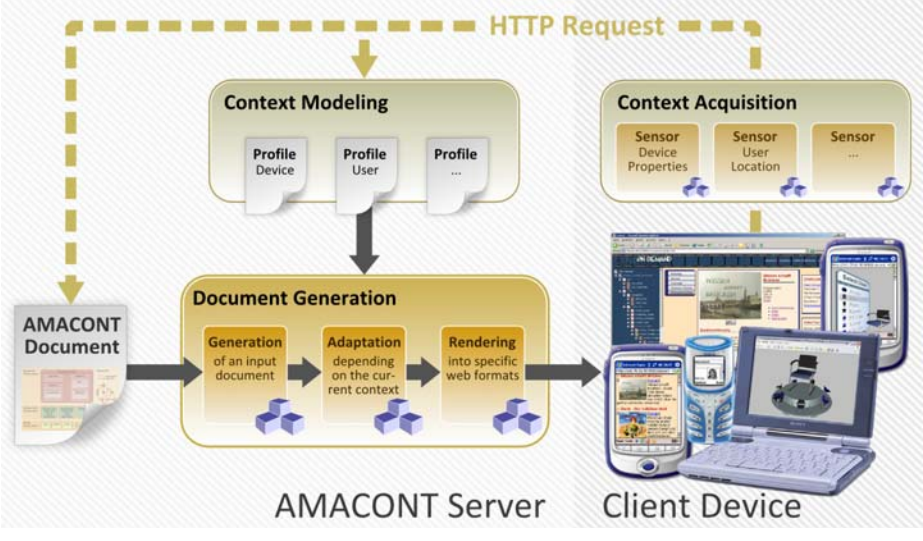

Fig. 3. The AMACONT Publication Process

2.4 Context Acquisition

Whenever a request is sent to an AMACONT web application, this request per se contains some context data on the client, e.g., preferred content language and used browser. As this little context information can hardly serve for rich adaptation, AMACONT deploys additional *sensor components* on the client, embedded as Java or JavaScript fragments into delivered pages. Sensor components can determine simple data like the available screen resolution or track more complex information like the user's interactions with the page, or his location.

2.5 Context Modeling

On the server, the gathered sensor information is processed by the *context modeling components*. An important component within the context modeling is the *user modeling* [6]. It provides user modeling strategies that require explicit feedback (e.g., filling out questionnaires) as well as an automatic modeling mechanism that allows an implicit analysis of user interactions. Context modeling also comprises *device modeling* components that process sensed device capabilities. In addition, AMACONT

also provides *location modeling* mechanisms that can be used to develop and deploy location-based services [7].

In order to be accessible by a broad range of web applications, AMACONT offers an extensible *context model*. It stores data in different context profiles. Each profile relies on CC/PP[2], an RDF grammar for describing device capabilities and user preferences in a standardized way. Context modeling and sensor components for user, device and location profiles have already been implemented and are in detail presented in [7]. In order to support the addition of further context modeling techniques, the framework provides generic extension mechanisms for adding new sensor components and context modelers.

2.6 Document Generation

Each requested document passes the publication pipeline where it is adapted according to the context model. To this end, elements are removed from the document containing all variants to eventually leave only the desired, adapted information. The last step of the pipeline carries out the transformation to a specific output format, such as XHTML or WML. The resulting document can then be delivered to the specific target device where it is rendered accordingly. For a detailed description of the adaptation pipeline, see [6].

3 Leveraging Aspect-Orientation for Modeling Adaptation

The variant-based adaptation concept in AMACONT presented in the previous section allows for a maximum flexibility, but reduces the overall reusability in the web application. As variants contain altered duplicates of the original component to cater for the special needs of users or devices, there is lots of duplicate content in the web application, which the author has to change consistently when making changes. Additionally, in an adaptive web application variants are spread all over the documents, missing some sort of connection that groups them by their goal (e.g., grouping all variants that cater for adaptation to the user's device). In order to maximize reusability and embrace the evolution of web applications, we extended AMACONT with a second concept for modeling adaptation: aspect-orientation. We argue that this allows us to separate the adaptation model from the rest of the application while offering an intuitive grouping mechanism for adaptation fragments.

3.1 Fundamentals of Aspect-Oriented Adaptation

Aspect-oriented programming (AOP) has been designed for "classic" desktop applications. The paradigm that AOP proposes is separation of concerns. Code that is spread throughout an application (i.e., logging or access restriction) is outsourced into isolated modules called *aspects*. These aspects are automatically merged with the *base application* (i.e., the application without the orthogonal concerns) at a later time, in a process called *weaving*.

[2] Composite Capability/Preference Profiles, http://www.w3.org/Mobile/CCPP/

3.2 Aspect-Orientation in Other Web Application Models

The idea of aspect orientation can already be found in UWE, where specific types of adaptation such as link hiding, link annotation and link generation were separated into different aspects ([1]). Also the UML-based WebML was redesigned with aspect-oriented features for the purpose of modeling ubiquitous web applications ([10]). AMACONT mainly differs from these approaches in that we transfer the AOP concept to the level of XML documents, allowing arbitrary transformations.

3.3 Aspect-Orientation in AMACONT

AMACONT employs traditional AOP concepts: the base is a regular web application, that is, typically without adaptivity. On top of such existing web applications, authors can instantiate *adaptation concerns* (Figure 4) that group adaptation on a requirement-level. Typical examples of adaptation concerns are *device independence*, *location awareness*, *internationalization* or *role-based access*.

Because concerns offer only a high level view, adaptation is further subdivided into *adaptation aspects*. These are medium-sized adaptation fragments that allow authors to impose an order on the weaving process, which is needed for a predictable result, as the weaving order generally plays a decisive role in AOP (e.g., see [9]).

Finally, the smallest possible piece of adaptation is an *advice*. Advices describe how to transform a given component within an AMACONT document. To this end, they may instantiate *adaptation patterns*, i.e., reusable transformations that resemble typical adaptation techniques like the ones identified by the hypermedia community ([3, 4]). The adaptation patterns available to the author differ, depending on the adaptation concern he is working on. We already have implemented a number of patterns, e.g., *first sentence elision*, *image rescaling* or *layout change* for instant use. Still, authors are free to define their very own patterns to achieve any adaptation effect they want. Advices also contain an adaptation precondition, similar to classic variants, that specifies the circumstances under which to perform the transformation. Finally, advices are connected to the web application via their *pointcut*. The pointcut, expressed in XPath, is a reference to one or many AMACONT components. Therefore, it is possible to easily apply one advice to an arbitrary number of components.

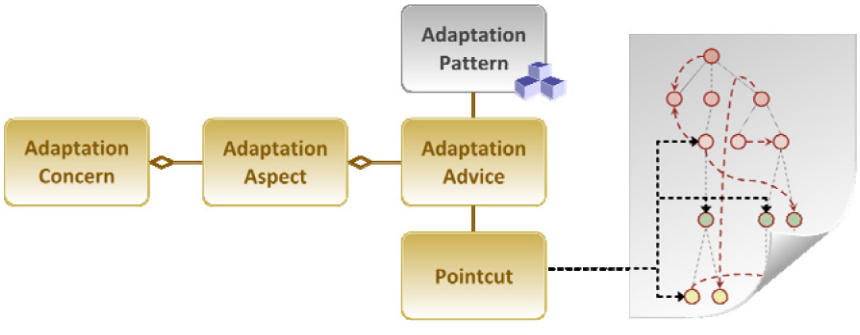

Fig. 4. Aspect Model of AMACONT Adaptation

An adaptation model based on aspects can be stored either for every AMACONT document or for the whole web application. The aspect weaver then has the task to merge this model with the base application.

3.4 Aspect Weaving in AMACONT

For weaving aspects, AMACONT offers two general choices: weaving at design time and weaving at run time.

Design time weaving offers one obvious advantage: Because it is a one-time process of generating alternative variants, it creates only a neglectable overhead. Then, at run time the server only needs to decide which (pre-made) variant to choose for a user. However, the greatest disadvantage of design time weaving is the inability to cover dynamic adaptation. For example, it is not possible to insert a user's name into the document, because that data is not known at design time.

In contrast, run time weaving shows opposite characteristics: Working with dynamic data is no longer a problem, while the downside is that it requires weaving at every request, thus reducing performance. But weaving at run time offers some more advantages: depending on the server load, it becomes possible to "deactivate" costly aspects temporarily. Moreover, it is possible to create alternative aspect configurations that are chosen dynamically based on the context. For example, authors could model three different levels of location awareness and base the selection on whether the user is a paying customer.

With aspect-oriented adaptation, web engineers are able to nicely separate adaptation from the rest of the web application. However, adaptation preconditions still show a semantic gap. While the AMACONT context model can only supply generic information on the user, his device and context, adaptation authors typically think in terms of higher granularity, e.g., "all returning customers from around Dresden". In the next two sections, we present a new approach for bridging this gap.

4 Semantics-Based Adaptation

One of the important issues in improving and making adaptation more sophisticated is to incorporate semantic web technology in AMACONT, allowing us to specify the meaning of the data by using meta-data and ontologies, and define their relationship. This enables applications to tie, exchange, and reuse data from different external data sources, including the data resources on the Web. Consequently, this is useful to reduce the knowledge load for adaptation conditions, as well as to simplify queries, or even use knowledge that is not available at design time. The semantic web technology also makes reasoning about the data by applications possible, allowing the semantic gap between data to be bridged. Therefore, by exploiting the semantic web technology, AMACONT is able to provide adaptation in a richer and more flexible way. And even though some work has been done on utilizing external knowledge for adaptation purposes (e.g., [8]), we do not know of another approach that applied external knowledge to simplify adaptation.

In order to make the semantic information of data accessible to the applications, they have to be represented in a machine-accessible format. W3C has released the

Resource Description Framework (RDF, cf. http://www.w3.org/RDF), a standard data model for representing meta-data and describing the semantics of information in a machine-accessible way. RDF defines statements about information resources in the form of subject-predicate-object expressions called *triples*. As the subject of one statement can be the object of another, a collection of statements forms a directed, labeled graph. There are several syntaxes to represent RDF, like an XML serialization, but obviously the interesting concept is the graph-based model that lies behind it. RDF statements can be stored in RDF repositories like Jena (cf. http://jena.sourceforge.net) or Sesame (cf. http://www.openrdf.org). For accessing information stored in RDF repositories, there are several RDF query languages (e.g., SeRQL [2] and SPARQL (cf. http://www.w3.org/TR/rdf-sparql-query)).

In this project, we use SPARQL as the RDF query language that is a W3C standard. As an example of a simple SPARQL query, consider the following:

```
PREFIX geo: <http://www.geography.org/schema.rdf>
SELECT ?capital ?country
WHERE { ?x   geo:cityname ?capital ;
             geo:isCapitalOf      ?y .
        ?y   geo:countryname      ?country ;
             geo:isInContinent    geo:Africa . }
```

The `PREFIX` clause is used to define namespace prefixes. The namespace `geo` is bound to the a URL that specifies geographical domain concepts. The `SELECT` clause lists the variables for which bindings will be returned. Note that variable names are always preceded with "?". The `WHERE` clause provides the graph patterns to be matched against the data graph. The ";" notation is used to list triples that start with the same subject. So, the above query looks for a city object `?x` with the name `?capital`. Furthermore, `?x` is the capital of country `?y`, and `?y` must have the name `?country` and be in the continent of Africa. Finally, it returns all combinations of `?capital` and `?country` for which these triples exist.

We now look at how we use a semantics-based adaptation extension in AMACONT to overcome granularity issues. Let us first look at the problem at hand, by looking at an example. Suppose we want to select the language in which we present pages in our application, based on where a user lives. In this way we could choose to give someone who lives in Germany a German version of the page, and other users the English version of the page. However, suppose we have a user model that contains the following location information about a user (in abbreviated TURTLE[3] syntax):

```
:userJohn    :livesin     :cityX
             a            :city;
             :cityname    "Dresden" .
```

In other words, we only know in this case that user "John" lives in a city called "Dresden". The question is now where we get the knowledge from to understand that Dresden lies in Germany so that we should give user "John" the German version of the page. One possible solution would be of course to extend the user model so that it

[3] Cf. http://www.w3.org/TeamSubmission/turtle/

also contains the information that user "John" also lives in the country of Germany, and ask the user for this information. However, this would bother the user with additional questions while we could already deduce the country from the city. Even though it might still be possible to ask the user in this concrete example, one can easily think of cases where this is not viable or even possible (e.g., if the information is not known at design time). Another possibility would be to extend the adaptation query with an enumeration of all German cities, which obviously is also not really viable as it would be extremely laborious to write such queries and their performance would suffer dramatically. We take another approach by exploiting the knowledge of RDF and OWL ontologies (cf. http://www.w3.org/2001/sw) to solve our granularity problem. Many of such ontologies are freely available on the Web and can be reused for our purposes.

We consider an example in which we add multilinguality via aspect orientation. Given that we have an application in default English with a set of text objects in English, we added translations of those objects in German with a derived name pattern de/%name%. We can now add the following aspect-oriented adaptation:

```
<aspect id="ML1" concern="Internationalization">
   <advice>
      <pointcut>
         <condition>
            <aada:Sparql>
              SELECT ?country
              WHERE {   $CurrentUser :livesin ?country
                        a :country ;
                        :name "Germany" . }
            </aada:Sparql>
         </condition>
         <target>
            <xpath>//AmaTextComponent</xpath>
         </target>
      </pointcut>
      <pattern id="ReplaceComponentByName">
         <parameter id="replacePattern">de/%name%</parameter>
      </pattern>
   </advice>
</aspect>
```

This aspect oriented adaptation condition expresses that if the user is from Germany (denoted by the SPARQL query to the user model), we want to replace all (English) text elements by their German translation (with path de/%text%). If the user is not from Germany we just present the default English text (and thus do not adapt anything). Please note here that $CurrentUser is an AMACONT variable (denoted by the $-sign) that is substituted at run time with the actual identifier for the current user.

We now have to deal with the fact that the `{:livesin ?country}` pattern is not in our user model. Therefore, we first select an appropriate ontology that allows making the semantic connection we need, in our case between a city and the country it resides in. A good candidate for this case is the GeoNames Ontology[4]. This ontology simply provides us with additional knowledge:

[4] Cf. http://www.geonames.org/ontology/

```
geo:6551127 geo:Name       "Dresden, Stadt" ;
            geo:inCountry  geo:DE .
```

In plain English this means that we have an object with id "6551127", which apparently represents the city of Dresden and has a direct property which connects it to Germany ("DE") via the geo:inCountry property. This is a very fortunate simple example, as every location within a country has this inCountry element. However, it could also be necessary to follow more complex paths. If we need more information about the country of Germany (e.g., its population), we would have to follow the complete chain

Germany » Saxony » Regierungsbezirk Dresden » Kreisfreie Stadt Dresden » Dresden, Stadt

to find the Germany URL, namely geo:2921044.

As we can via this ontology make the connection between Dresden and Germany, we can conclude that the user lives in Dresden and *therefore* in Germany. In order to achieve that, we of course need to do some configuration in AMACONT.

We first need to align the location name in our local ontology (e.g., "Dresden") with the location name in the geo ontology (e.g., "Dresden, Stadt"). If these concepts are identical (for instance, by design), then no further configuration is needed in AMACONT. Otherwise, we can make a manual alignment for every user, or a (semi-) automatic alignment. The last can be configured by specifying two SPARQL queries.

We then first specify the following construct in our AMACONT configuration to indicate an alignment step:

```
:align [ sourceConceptQuery "query1 details" ;
         targetConceptQuery "query2 details" ; ]
```

Now we fill in the query details. For the source concept we specify which concept in the UM we want to connect with a target concept of our specialized ontology. In simple cases we allow for simple declarations like for "query1" in this case:

```
?city a um:City
```

Here we specify that we want to select all concepts of type city of the UM vocabulary, i.e., the only variable. For "query2", the target concepts, we specify a (somewhat simplified) SPARQL query of how to find the aligned city in the geo ontology. In this query we can align the result by using it for query1, referring to it by the variable $city. The "query2" could then look like this:

```
SELECT ?feature
WHERE {
          ?feature    a                 geo:Feature ;
                      geo:featureCode   geo:P.PPL ;
                      geo:name          ?name ;
                      geo:population    ?population .

          FILTER (regex(?name, $city))
       }
ORDER BY DESC(?population)
```

In this query we are looking for a "feature" (in GeoNames terminology) that has the featureCode geo:P.PPL (code for city) and the name should contain the original city name indicated by the AMACONT variable $city in the FILTER clause. Note that we assume that the target location here contains the city name of the source as a substring (i.e., this might be a limitation for more complex searches that cannot be expressed like this). We also include "?population" in our search to solve ambiguity issues. Suppose that there are more cities in the world that are called Dresden (actually six such cities/towns exist in the USA), so we have to find a heuristic to determine the most probable target concept. For our example, we use the heuristic that the most populated city is the most probable one. We order the results by population and, in the case of more than one result, we use the first result as the target concept.

We have now aligned concepts in our user model to concepts in our helping ontology. After alignment we want to specify a reasoning rule that defines how knowledge in the helping ontology extends the knowledge in our user model. In our case, we define an entailment rule that specifies that if a user lives in a city, he also lives in the country in which that city is located. This rule is basically of the form:

$$\left.\begin{array}{c} <?X\ \text{:livesin}\ ?Y> \\ <?Y\ \text{rdf:type}\ \text{:city}> \\ <?Y\ \text{geo:inCountry}\ ?Z> \end{array}\right\} \Rightarrow \left\{\begin{array}{l} <?X\ \text{:livesin}\ ?Z> \\ <?Z\ \text{rdf:type}\ \text{:country}> \end{array}\right.$$

This (syntactically simplified) inference rule specifies that if X lives in Y of type city, then X also lives in the country in which Y is located. By applying this entailment rule, we can deduce that John who lives in Dresden, also lives in Germany and thus should get the German welcome page. With this configuration, AMACONT can now effectively use the external ontology to solve granularity issues in our user model.

5 Implementation

Both the proposed aspect-oriented modeling approach and the semantics-based adaptation have been implemented in AMACONT's publication pipeline, enabling authors to make use of the new features.

For aspect orientation, we extended the document generation pipeline by adding another transformation component before the adaptation transformer (Figure 5, green parts have been added/extended). This way, we can weave adaptation aspects at runtime, but still allow authors to use the more generic static variants, which are then processed by the existing adaptation transformer.

The aspect weaver takes as input an AMACONT document together with an adaptation description. It then tries to match the advice's pointcuts as specified in the adaptation description with the components in the input document. The following code snippet shows an extract from the aspect-oriented adaptation definition, resizing all images in the navigation bar if the user accesses the application with his PDA.

```
<aspect id="RI1" concern="ResolutionIndependence">
   <advice>
      <pointcut>
         <condition>
            <adaptationclass>Device_PDA</adaptationclass>
         </condition>
         <target>
            <xpath>//*[@id=”nav”]/aco:AmaImageComponent</xpath>
         </target>
      </pointcut>
      <pattern id="ReduceImageSizeByTranscoding">
         <parameter id="ratio">0.5</parameter>
      </pattern>
   </advice>
</aspect>
```

If a match is found, the advice's transformation is executed, modifying the affected components. Advices are executed one after another, depending on the order specified by the author.

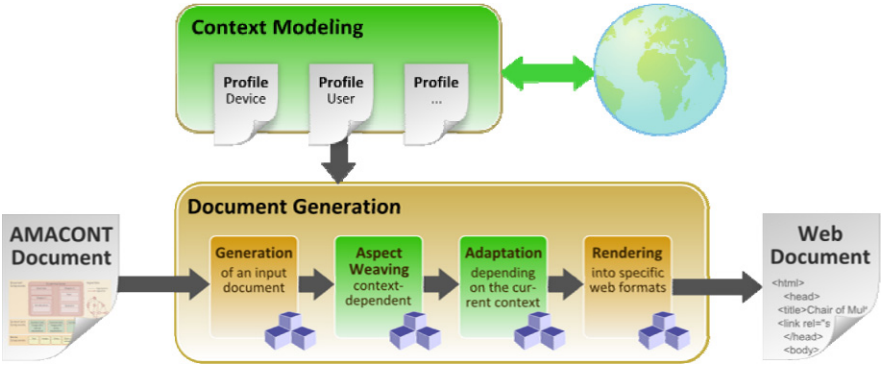

Fig. 5. The Extended AMACONT Document Generation Pipeline

In order to add the new semantic web capabilities, we have chosen to base our implementation on the open-source RDF framework Sesame[5]. Sesame offers a good level of abstraction on connecting to and querying of RDF storages, similar to JDBC.

We have extended both the existing adaptation transformer and the new dynamic weaver to not only support terms that rely on the local context model, but also to process SPARQL statements. These statements are evaluated at runtime, replacing AMACONT variables and finally querying the corresponding SPARQL endpoint.

Queries can be added to conditions of both variants and adaptation advices by encapsulating them via the new `Sparql` element in an `If` term:

[5] http://www.openrdf.org/

```
<aada:Sparql>
   SELECT ?conceptVisits
   WHERE { $CurrentUser   :livesin?country
                          a         :country ;
                          :name     "Germany" . }
</aada:Sparql>
```

The SPARQL syntax itself is not touched, just extended with the notion of AMACONT variables (denoted with a preceding $), which are replaced before running the query.

6 Conclusion and Future Work

In this paper we have presented methods for extending an existing web application framework with orthogonal facilities for adding adaptivity and for extending the user context modeling with semantics. To this end, we have applied aspect-oriented techniques to the modeling of adaptivity on the one hand and concepts and technologies from the semantic web for building and accessing a rich context on the other hand. We have illustrated this in terms of AMACONT, but these techniques are suitable for enhancing web application frameworks in general.

As a continuation of our work, we plan to further combine both worlds by allowing authors to use semantic web techniques for specifying the content of a web application. An example use case for this is displaying Wikipedia pages of local sights. By using an ontology, sights within a greater area around the user's location (not only within the same city) can be recommended in a tourist guide. Aspect-orientation can then be used to add adaptation on top of such content, providing support for orthogonal adaptivity on all three application layers: content, structure and presentation. Another point is evaluation of the system's performance, especially measuring scalability. In our implementation we found no performance issues given that we use a relatively straightforward application and mid-sized ontology. However, as Semantic Web reasoning applications typically run into scalability issues, we want to explore the scalability limits of our approach. Finally, we plan to extend our authoring tool, AmaArchitect, to provide authors with means for dealing with the new semantic context. Authors could thus browse datasources and select from the concepts and relations available there.

Acknowledgements. Part of this work is funded by the German Research Foundation (DFG) within the project "HyperAdapt".

References

[1] Baumeister, H., Knapp, A., Koch, N., Zhang, G.: Modelling adaptivity with aspects. In: Lowe, D.G., Gaedke, M. (eds.) ICWE 2005. LNCS, vol. 3579, pp. 406–416. Springer, Heidelberg (2005)

[2] Broekstra, J., Kampman, A.: An RDF Query and Transformation Language. Springer, Heidelberg (2006)

[3] Brusilovsky, P.: Adaptive navigation support. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) Adaptive Web 2007. LNCS, vol. 4321, pp. 263–290. Springer, Heidelberg (2007)

[4] Bunt, A., Carenini, G., Conati, C.: Adaptive content presentation for the web. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) Adaptive Web 2007. LNCS, vol. 4321, pp. 263–290. Springer, Heidelberg (2007)
[5] Fiala, Z., Hinz, M., Meißner, K., Wehner, F.: A component-based approach for adaptive dynamic web documents. Journal of Web Engineering 2(1&2), 058–073 (2003)
[6] Hinz, M., Fiala, Z.: Amacont: A system architecture for adaptive multimedia web applications. In: Workshop XML Technologien für das Semantic Web (XSW 2004), Berliner XML Tage (October 2004)
[7] Hinz, M., Pietschmann, S., Fiala, Z.: A framework for context modeling in adaptive web applications. IADIS International Journal of WWW/Internet 5(1) (June 2007)
[8] Krištofič, A., Bieliková, M.: Improving adaptation in web-based educational hypermedia by means of knowledge discovery. In: HYPERTEXT 2005: Proceedings of the sixteenth ACM conference on Hypertext and hypermedia, pp. 184–192. ACM, New York (2005)
[9] Nagy, I., Bergmans, L., Aksit, M.: Composing aspects at shared join points. In: NODe/GSEM, pp. 19–38 (2005)
[10] Schauerhuber, A., Wimmer, M., Schwinger, W., Kapsammer, E., Retschitzegger, W.: Aspect-oriented modeling of ubiquitous web applications: The aspectwebml approach. In: Engineering of Computer-Based Systems, 2007. ECBS 2007, Tucson, Arizona. 14th Annual IEEE Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, pp. 569–576. IEEE Computer Society, Los Alamitos (2007)
[11] Schwinger, W., Retschitzegger, W., Schauerhuber, A., Kappel, G., Wimmer, M., Pröll, B., Cachero Castro, C., Casteleyn, S., De Troyer, O., Fraternali, P., Garrigos, I., Garzotto, F., Ginige, A., Houben, G.J., Koch, N., Moreno, N., Pastor, O., Paolini, P., Pelechano Ferragud, V., Rossi, G., Schwabe, D., Tisi, M., Vallecillo, A., van der Sluijs, K., Zhang, G.: A survey on web modeling approaches for ubiquitous web applications. International Journal of Web Information Systems 4(3), 234–305 (2008)