Building Self-Managing Web Information Systems from Generic Components

Geert-Jan Houben[1], Zoltán Fiala[2], Kees van der Sluijs[1], Michael Hinz[2]

[1] Technische Universiteit Eindhoven
PO Box 513, NL 5600 MB, Eindhoven, The Netherlands
{g.j.houben, k.a.m.sluijs}@tue.nl
[2] Technische Universität Dresden
Mommsenstr. 13, D-01062, Dresden, Germany
{zoltan.fiala, mh5}@inf.tu-dresden.de

Abstract. The increasing need for device independence and personalization forces organizations to automatically adapt their Web Information Systems (WISs) to individual users and their client platforms. Moreover, the Web's evolution to a dynamic interaction medium also requires such WISs to be self-adaptive, i.e. to dynamically adjust themselves to a constantly changing reality. Still, there are no generic software components to manage this multitude of adaptation aspects. This paper presents the Self-Adapting GAC (SAG), a generic component for realizing self-managing adaptive WISs. It has the capability to perform adaptive data transformations, but it can also dynamically reconfigure itself to the current situation. For configuring the SAG an RDF-based rule language is introduced, providing rules for both content adaptation and self-configuration. Moreover, based on the modular document generation architecture of the AMACONT project an implementation of the SAG is provided. The resulting architecture is explained in detail and elucidated by a running example prototype.

1 Introduction

Organizations in search for effective support from their information systems are faced with an increasing complexity involved in the combination of software and information. With the evolution of the Web, the development of Web-based information systems showed an even bigger demand for ways to control the design of the systems. The Web features a key step of standardizing software, or at least parts of it. Standards like HTTP and tools like Web browsers reduce most of the engineering of Web applications to the modeling of the content in terms of languages like HTML.

However, this rather simple mechanism of the "surface Web" does not suffice for the professional information systems that get their content from the "deep Web", the Web that retrieves its content from strongly structured data repositories such as databases. These Web Information Systems (WISs) [1] are generally characterized by data transformations: the content from the sources available via

the Web is transformed by the WIS into nice presentations in HTML or similar formats. Providing functionality on the basis of such data transformations leads to concepts like adaptation: in order to get specific functionality the generic transformation is adapted to the situation. Many approaches for adaptation use tailor-made solutions, but there are also generic approaches available [2, 3]. In recent work we introduced the Generic Adaptation Component (GAC) [4], a software component that with a little configuration produces the right data transformations (adaptations).

In this paper we consider an extension of this standard software component so that besides adaptive data transformation it can also adapt itself to the situation by changing its own configuration. With this new component, called Self-Adapting GAC (SAG), we are able to easily construct the complex functionality of WISs that can configure themselves in a dynamic environment.

First, in Section 2 we extensively discuss the role of adaptation in an Adaptive WIS (AWIS) and make the distinction between static and dynamic adaptation. Then, in Section 3 we introduce SAG. We show that it is a generic component being able to do transformation given some configuration (e.g. some describing model). Furthermore we show that it can configure itself by maintaining a dynamic context model. After that, based on the modular document generation architecture of the AMACONT project [5], we discuss the implementation in Section 4. There, we describe how to create an AWIS by placing several SAGs in a line. Furthermore, we expose a number of implementation details and a link to the actual running prototype. Throughout the paper we use a concrete application as a running example to help the reader understand the main concepts.

2 Adaptation in WIS architectures

Creating an Adaptive WIS is a complex task requiring to consider a multitude of design aspects. It is therefore beneficial to extract the common functionality of these applications, such as navigation, adaptation and user interaction. Frameworks that enable the generation of AWISs by configuring a limited number of models can greatly simplify the life of developers and give them more control. Furthermore, by providing them with configuration and adaptation primitives they are able to generate an AWIS that can adapt itself to a changing context.

2.1 Adaptation Types

Current AWIS frameworks like Hera [6], OOHDM [7], SHDM [8], WebML [9], WSDM [10] etc. partition AWISs into a number of separate layers. In this way they try to reduce complexity and make the AWIS controllable. A typical partition is by discerning content, navigation, and presentation. The content structure describes the input in term of concepts and the relationships between these concepts. For example, this layer gives room to integrate data from different data sources. In the navigation layer the content is semantically structured with relationships specific for the application (navigation, interaction) setting in which

the WIS is used; for instance based on a business model. A typical example of such a relationship structure is the navigation structure for the browser. Within the navigation the semantic relationships describe the content elements that are on a page and how these connect to content on other pages. In the presentation layer the presentation layout of the presentation pages is described. Examples of this are defining style elements, sorting order and placement [11].

Within these layers the notion of adaptation can be extended. Based on [12], we distinguish different kinds of adaptation, namely adaptability and adaptivity to the person and the context. Adaptability (or static adaptation) means that the generation process is based on available information describing the situation in which the user will use the presentation, e.g. color depth and layout preferences. Adaptivity (dynamic adaptation) is the adaptation included in the generated adaptive hypermedia presentation. It means that all layers within the application are dynamically affected by the user's knowledge. The user's knowledge is determined by interpreting it's interaction behavior, i.e. following links and entering data. Possible examples of adaptability and adaptivity across the design levels are summarized in Table 1.

	Adaptability	**Adaptivity**
Content level	image adaptation to color depth of browser	media adaptation to fluctuating bandwidth
Navigation level	hiding specific links for unauthorized users	link sorting according to user's changing knowledge
Presentation level	CSS adaptation to user's layout preferences	layout rearrangement when resizing the browser's window

Table 1. Adaptability/adaptivity examples across the design levels

2.2 Data Transformations for Presentation Generation

While we see different types of adaptation, in practice most of the adaptation is realized through data transformations. Languages like XSLT or Java that support the transformation of content data in formats like XML and HTML can be used to express how the content is adapted to the context. The adaptation of the content to the context is in fact realized through parameterization of the data transformations. The models used in methods such as Hera become parameters that drive the data transformations: while the models for the content describe the content that needs to be transformed, the models for the adaptation describe how that transformation should be done in this context.

With this we can generate a presentation that best suits the situation of the user and the context, and we can change the presentation while being used. These types of adaptation lead to the use of two additional models for the adaptation; one for user modeling and one for modeling the usage context. The latter is used

to model conditions that determine the specified behavior of the presentation. These two models are dynamic in the sense that they depend on the user and can get updated during browsing. In order to complete the application-user interaction support, there is a need for a data feedback mechanism. This allows user input for the underlying application and a more extensive mechanism for changing the user model and context.

2.3 Generic Adaptation Components

Most currently used AWISs are proprietary solutions serving a specific application scenario from a specific domain. They are developed by experts being responsible for the complete design, implementation, configuration and deployment of those systems. Recently emerged frameworks for AWIS design [6, 7, 8, 9] assume to develop AWISs "from scratch" by using rich design models in which the adaptation is embedded. These frameworks involve quite some predefined different steps and every step comes with a rather steep learning curve.

From the point of view of the software, there is however a big improvement possible. As all steps involve similar data transformations, it would be preferable to have a generic component that can be used in all steps. In this way the learning curves may be less steep. Furthermore, developers can thus lace their own AWIS framework, applicable to their specific applications and methodology. As already mentioned, we recently introduced the Generic Adaptation Component (GAC) [4] aiming at adding adaptation to WISs.

However, integrating rich kinds of user action with the presented data introduces considerable complexity in such an adaptation component. Typically, standard adaptation implementations do not provide sufficient support for developers that aim at adding user interaction to their existing applications that goes beyond the traditional link-following. In those cases the capability of the component to receive feedback from the presentation and react to that is a must to enable the more advanced user interaction. At the same time, this additional capability allows more possibilities to influence the adaptation component's configuration. In this way, the adaptation component moves from a rather modest transformer of content to a more self-controlled provider of information.

This paper's topics aim at configuring dynamic Web applications. We deal with questions like how to extend the capability to adapt to a level of self-configuration and how to exploit generic software components to manage the multitude of adaptation aspects?

We illustrate our solutions with the running example used throughout this paper [13]. The example is a dynamic WIS providing information about our research project called "Hera meets AMACONT!". The project is described by a set of attributes, such as a name and a textual description. There are members working at the project, each characterized by a name, a CV, a picture, as well as some contact data. They produce publications on their research efforts, which are described by a title, the name of the corresponding conference or journal, and the year of publication.

3 A Generic Component for Building Adaptive WIS

In this Section we present the SAG, a generic software component for adaptation to data transformations with the capacity to change its own configuration. Figure 1 shows how SAG is embedded in the overall Web infrastructure. The component receives and processes XML-based content, for example provided by a Web application content generator or Web server. It adjusts and outputs this content according to the context, for example the preferences and properties of individual users and their client devices. In return for the output it receives interaction requests from clients, such as page requests or queries etc. Based on this requests it will perform corresponding actions. Thus, the component can provide a Web interface to take care of the application-user interaction in which the application sends XML data to the user and receives back corresponding requests.

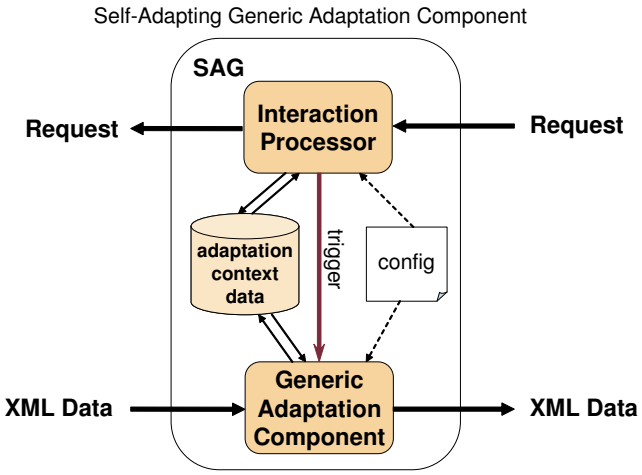

Fig. 1. SAG overview.

The basic data transformation (adaptation) part of the SAG is based on our Generic Adaptation Component (GAC [4]), a transcoding tool aiming at dynamically adapting XML-based Web input to the current adaptation context data. Being generic, the GAC can perform different adaptations on its input, the recipe for which is specified by its *configuration*. This configuration consists of a set of *adaptation rules*, each dictating a different content adaptation aspect. To take the current state of the user and device context into account, adaptation rules can reference arbitrary parameters from the *adaptation context data*. In order to support adaptivity, the configuration also contains *update rules* allowing to manipulate this context data according to the user's interaction history.

On top of the GAC's basic data transformations (in Figure 1 the data flow from left to right) the SAG is also capable of processing interaction requests,

depicted by the flow from right to left. These requests comprise all information received from the client side, such as page requests, queries, data sent by submitting forms etc. The SAG can serve (parts) of these requests on its own, and/or redirect them to other processing units in a possibly modified way. In terms of Figure 1 the former means that the component sends output data to the right, while the latter means that a request is passed on to the left. As possible processing actions we mention the updating of the adaptation context data or the re-triggering of the data transformation process. The recipe for these actions is also described in the configuration of the SAG by means of the *request processing rules*.

3.1 WIS Composition

While the scenario shown in Figure 1 utilizes only one SAG, it is possible (and advisable) to employ several independent SAGs at different stages of a WIS generation process. Consider an example in which XML data gets adjusted to the user for personalization, the adjusted XML gets transformed to HTML, and subsequently the HTML data gets adjusted to the presentation capabilities of a PDA. This overall process can be viewed as a sequence of transformations. As described in Section 2, a WIS can be efficiently realized with three layers, each responsible for its own specific adaptation processes. Thus we observe, how a typical AWIS can be put together from generic SAGs, each realizing one of its transformation layers (see Figure 2).

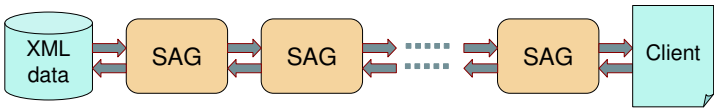

Fig. 2. SAG composition

In Section 3.2 we first show how the basic data transformations (i.e. the flow from left to right) are adapted. Then, in Section 3.3 we show how the feedback (from right to left) is processed, thus realizing the changes to the adaptation configuration.

3.2 Adaptation Configuration

We now consider in more detail the configuration of the SAG. As we said earlier, the data transformation part of the SAG is based on the GAC. It is controlled by its RDF-based configuration that consists of a set of *rules* specifying the content units to be adjusted, the adaptations to be performed on them, and (in the case of adaptivity) the way the adaptation context data has to be updated.

A graphical excerpt of the RDF schema defining our rule hierarchy is depicted in Figure 3. The top of this hierarchy is the abstract class *Rule*. A *Rule*

is always bound to a *Condition*, i.e. it is activated if and only if that condition holds. A *Condition* is an arbitrary complex Boolean expression referencing parameters from the adaptation context data. Rules can be either *Adaptation Rules* or *Update Rules*.

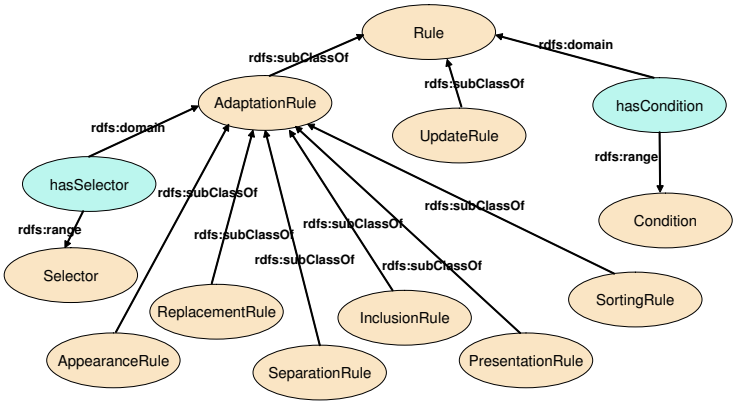

Fig. 3. GAC Rule Schema.

Adaptation Rules describe adaptation operations to be performed on specific parts or structures of the input content. They have a *selector* property that contains an XPath expression for unequivocally identifying these parts. Whenever there are several rules addressing the same content element, they are ordered by their *priority* property.

In order to use our Adaptation Rules (and the corresponding operations) in different application scenarios, we decided to identify a common set of generally applicable rule primitives. Based on Brusilovsky's survey on basic methods and techniques for content and navigation adaptation [14], as well as our recent work on presentation adaptation [11], we selected and implemented the following rules[3]:

An *Appearance Rule* realizes one of the most basic adaptation methods: the selected content is included in the output only if the associated condition is valid. The following example omits images (in Hera referred as `Slice.member.picture`) for devices being unable to display them. Note that adaptation context data parameters are denoted with a $ sign.

```
<gac:AppearanceRule gac:selector="//Slice.member.picture">
  <gac:Condition gac:when="($ImageCapable==yes)"/>
</gac:AppearanceRule>
```

[3] In the rule illustrations from our concrete running example, the XML data happens to be organized in terms of slices, which are the navigation primitives discerned in Hera; we point out that the rules are however generally applicable.

Separation Rules are a variation and less strict modification of *Appearance Rules*. Instead of being elided, content units with invalid conditions are put to a separate page and replaced by a link to that page.

An *Inclusion Rule* realizes the inverse mechanism. If the condition is valid, external content referenced by an URL is included in the output document at the place determined by the selector. Note that an additional property (*what*) is used to specify that URL. In our prototype application we use a number of *Inclusion Rules* to show additional information for users being interested in details. Following code snippet inserts an additional paragraph to the project's textual description.

```
<gac:InclusionRule gac:selector="//Slice.project.description]">
  <gac:Condition gac:when="$InterestedInDetails=yes"/>
  <gac:what>http://www-mmt.inf.tu-dresden.de/projectdetails</gac:what>
</gac:InclusionRule>
```

Note that the fact whether a user is interested in details is determined by the number of his page visits. This mechanism will be described in detail in Section 4.4.

A *Replacement Rule* substitutes specific XML fragments with alternative fragments.

Link Wrapper Rules can be effectively utilized to manipulate hyperlinks. Their main application in the SAG is the modification of hyperlink targets encountered in the input XML documents. In this way users' clicks on the generated links are fed back to the SAG and processed by its own interaction processor module instead of just being passed back to the server of the link (see Section 3.3). Furthermore, according to the adaptation context data the link wrapper rules can also add additional (personalized) request parameters to hyperlinks.

Following *Link Wrapper Rule* modifies all hyperlink targets to be redirected to a specific SAG component (when more SAGs are used). Moreover, it also adds a new parameter representing the name of the current user to them. Note that the original link target is automatically attached to the new URL in the form of an additional parameter called *oldURL*.

```
<gac:LinkWrapperRule gac:selector="//A">
  <gac:Condition gac:when="($visuallyimpaired==yes)"/>
  <gac:toURL>http://my-sag.org</gac:toURL>
  <gac:param name="UID">$UserName</gac:param>
</gac:LinkWrapperRule>
```

As an example, when applied to the specific user with the ID called *myUID* and the original URL `http://myoldurl.org`, the above rule creates following link target:

```
http://my-sag.org?UID=myUID&oldURL=http://myoldurl.org
```

Note that *Link Wrapper Rules* can not only be applied to hyperlink targets but to arbitrary URLs, e.g. also to the *action* attributes of Web forms. The way how the SAG processes such requests will be described in detail in Section 3.3.

Whereas the rules mentioned above address single content units, there are also rules adapting sets of content units, such as all child elements or all variants of a specific content unit. A *Select Rule* selects one of the available content units according to a selection function. *Paginator Rule*s aim at dividing sets of content units in a number of subsets, each containing a predefined number of elements. Finally, *Sorting Rules* aim at ordering the selected content units according to one of their attributes.

In our running example the list of project members shown on the project homepage is sorted according to whether the user already saw their homepages. Members the user was already interested in are shown in the beginning of the list. As no condition is defined, this rule is always executed. Note that this is an example of adaptivity. The corresponding rule for updating the adaptation context data will be shown later.

```
<gac:SortingRule gac:selector="//Slice.project.members">
  <gac:by>$SliceVisited[@ID]</gac:by>
  <gac:order>desc</gac:order>
</gac:SortingRule>
```

Finally, a set of *Presentation Rules* have been created. In contrast to other *Adaptation Rules*, they aim at transforming device-independent XML input to a concrete Web implementation format, such as HTML, cHTML or WML. Based on previous results of the AMACONT project [11], *Presentation Rules* can assign so called *layout managers* to such groups of data containers (slices in our example). Assumed that a desktop PC with sufficient horizontal resolution is used, the following rule arranges the pictures of project members in a tabular way. The *cols* attribute determines the number of columns in that table.

```
<gac:PresentationRule gac:selector="//Slice.member.picture">
  <gac:Condition gac:when="($InnerSizeX>600)"/>
  <gac:layout gac:type="GridLayout">
     <cols>3</cols>
  </gac:layout>
</gac:PresentationRule>
```

Update Rules aim at updating the adaptation context data, and thus keeping track of history. They are used to change (or create new) context parameters and are triggered whenever the GAC processes an input document. The action performed by an Update Rule is specified in its *do* property. The *phase* property determines whether the rule is executed before (*pre*) or after (*post*) the transcoding process[4].

Among the Adaptation Rules of our running example a Sorting Rule rule supporting adaptivity was mentioned. It requires to keep track of the slices that have already been visited by the user. This mechanism is supported by the following Update Rule. The XPath expression identifies all slice instances.

[4] Introduced by the AHAM reference model for adaptive hypermedia applications [15], the phase attribute is widely used in systems supporting adaptivity.

```
<gac:UpdateRule selector="//Slice">
  <gac:do>$SliceVisited[@ID]=true</gac:do>
  <gac:phase>post</gac:phase>
</gac:UpdateRule>
```

3.3 Feedback and Interaction Processing

As already mentioned, the SAG extends the GAC with the functionality of an *Interaction Processor*, which is responsible for the processing of interaction requests that come from the client side or another SAG. Depending on the embedding of the SAG in the entire system, these requests can originate from user interactions (such as following links or filling out forms) or from requests generated by other SAGs (see Section 3.1). With this interaction processing the component provides mechanisms for full application-user interaction, including dynamics in the adaptation.

The way the SAG processes requests is described in its configuration by means of so called *Request Processing Rules*. Parametrized by the received request parameters as well as the current state of the adaptation context data, they unequivocally define which actions have to be performed. Firstly, it is possible to dynamically update the adaptation context data according to new information received from the client side. Secondly, the SAG can also decide whether the request has to be served by itself or redirected (possibly in a modified way) to a different server component, such as another Web application server or the preceding SAG in the SAG pipeline (see Section 3.1).

In our running example users can directly influence the look-and-feel of the Web application by choosing different layout types from a select-list. The currently chosen layout is sent back to the SAG by means of a Web form. Following Request Processing Rule (triggered by the client request) changes the layout preference attribute of the user in the adaptation context data. Note that request parameters are denoted in the form $\$reqparam[x]$.

```
<sag:RequestProcessingRule>
  <sag:Condition gac:when="$reqparam[layout]!=''"/>
  <sag:do>$LayoutPref=$reqparam[layout]</sag:do>
</sag:RequestProcessingRule>
```

After executing possible update rules, the interaction processor of the SAG decides if the request has to be redirected. As in our concrete example the change of the user's layout preferences affects only the current SAG responsible for the presentation layer, it can re-trigger the data transformation process according to the new context data.

```
<sag:RequestProcessingRule>
  <sag:Condition gac:when="$reqparam[layout]!=''"/>
  <sag:do>retrigger</sag:do>
</sag:RequestProcessingRule>
```

Still, there are also cases when the SAG can not process (parts of) a request on its own. In our running example project members are allowed to add new publications to their Web pages by filling out and submitting a corresponding Web form. Since this data has to be written back to the original content store, it has to be handed back through all SAGs to the server that controls the content storage. Therefore, the following *Request Processing Rule* creates a new request consisting of a new URL prefix (addressing the preceding SAG) and a list of parameters describing the new publication.

```
<sag:RequestProcessingRule>
  <sag:Condition gac:when="$reqparam[pubname]!=''"/>
  <sag:redirect>http://my-sag2.org</gac:redirect>
  <sag:param name="pubname">$reqparam[pubname]</sag:param>
  <sag:param name="pubyear">$reqparam[pubyear]</sag:param>
  ...
</sag:RequestProcessingRule>
```

The rules executed by the SAG are parameterized by the *adaptation context data* containing up-to-date information on both the user and his browser device (client). Note that already the rules from the GAC were parameterized in this way, but for the request processing rules the role of the adaptation context data is even more interesting: in this data the interaction processor can store its temporal state information. The modeling of user and device in the application context data relies on CC/PP, an RDF grammar for describing device capabilities and user preferences [16]. In the style of the WAP User Agent Profile (UAProf) [17], a CC/PP-vocabulary for describing WAP devices in a standardized way, we have developed specific extensions to describe a large diversity of devices and user preferences.

4 Implementation

The SAG was realized with the aid of the AMACONT project's modular document generation architecture [5]. As illustrated in Figure 4, it consists of a configurable series of data transformation modules (GACs) and their corresponding interaction processors. Sent to this pipeline, XML documents are adjusted to the current user and device context in a stepwise way. Furthermore, the interactions gathered from the client side are processed (or redirected) by the coresponding context modeling modules. The document generator was realized based on the Cocoon publication framework.

4.1 Data Transformations

Within the SAG, the GAC was implemented as a custom Cocoon transformer in Java. It communicates with the adaptation context data repository and performs the appropriate data transformations on the DOM view of its input documents. In the current implementation the repository was realized based on the open

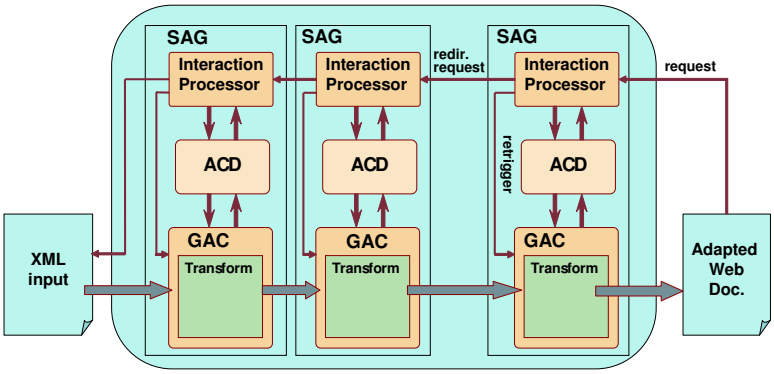

Fig. 4. AMACONT Document Generation Pipeline.

source RDF database Sesame. The GAC utilizes SeRQL (Sesame RDF Query language) for retrieving or updating this data. In order to effectively realize *Adaptation Rules* and *Update Rules*, a Java class was implemented for each rule type introduced in Section 3.2. Parameterized by rule-specific properties, they are optimized for performing the corresponding operations on the DOM-based representation of the input documents.

4.2 Realizing Presentation Adaptation

As already mentioned, *Presentation Rules* rules differ from other Adaptation Rules. By making use of AMACONT's layout manager concept they aim at transforming the input content to concrete output formats, such as XHTML, cHTML or WML. This transformation is performed by means of format-specific stylesheets written in XSLT. A detailed description of presentation layer transformations was given in [11].

4.3 Feedback Processing

As mentioned above, hyperlinks and Web forms included in the generated Web presentation can be redirected to the SAGs by means of Link Wrapper Rules. Whenever such a hyperlink or form is activated, the generation architecture of AMACONT [18] also allows for automatically gathering and submitting up-to-date information about users' device capabilities (e.g. browser type, current window size etc.) to the server side. Thus, all information describing the user's dynamic browsing behavior and client-side usage context is transferred to the corresponding SAG via HTTP requests.

The interaction processor module of the SAG was also implemented in Java[5]. According to the corresponding rule configuration, it firstly performs updates on

[5] By defining clearly defined interfaces, the modular architecture of AMACONT allows for utilizing arbitrary context modeling components for reacting on client requests and user interactions on the server side.

the adaptation context data by executing queries to the context repository. Then, it decides whether to re-trigger the data transformation process or redirect the (optionally modified) request to the preceding SAG.

4.4 Running Example Scenario

For illustrating the main functions provided by the SAG, we implemented a prototype application realizing the running example described in this paper [13]. As shown in Figure 4, it is composed of three SAGs realizing the content, the navigation and the presentation layers of the application. Note that this three-part architecture is only specific for the running example, the modularity of the AMACONT pipeline allows for arbitrary configurations.

Figure 5 shows two versions of the generated project homepage, one for a desktop PC and another one for a PDA. As dictated by our *Presentation Rule*, the limited horizontal resolution of the PDA does not allow to arrange the project members' photos in a tabular way.

Fig. 5. Generated Presentation.

When a user selects his preferred presentation style from the select list shown on the project homepage, this information is submitted to the third SAG responsible for the presentation layer. As this request does not affect the other layers (see the first two *Request Processing Rules* described in Section 3.3), it can be served by that SAG on its own. First, it reconfigures itself by updating the adaptation context data with the new information describing the user's lay-

out preferences. Then, according to the update context, it retriggers the data generation process (see the *retrigger* arrow in Figure 4).

Clicking on a project member's photo one can navigate to his personal homepage. During page generation the *Update Rule* defined in Section 3.2 is executed, indicating that the appropriate page (slice) was visited. Whenever a project member is viewing his own homepage, he can also add new publications by filling out a Web form. As this request contains a parameter called $\$pubname$, it can not be handled by the last SAG and is therefore fed back to the first SAG realizing the content layer in form of a *redirected request* (see Figure 4). Finally, according to our running example's *Sorting Rule*, the list of project members is dynamically reordered when navigating back to the project homepage.

In order to demonstrate self-configuration, an adaptation mechanism based on the observation of users' pages visits throughout several sessions was implemented. If the number of a user's page requests (slice requests) exceeds the average page request rate by a predefined percentage, the SAG automatically classifies him as "interested" by setting the corresponding adaptation context data parameter ($\$InterestedInDetails$) to *true*. As this change activates the *Inclusion Rules* aiming at showing additional information (see Section 3.2), the SAG automatically reconfigures its data transformation (adaptation) behavior. Thus, according to the users' long-term browsing behavior (groups of) rules can be added to (or removed from) the data transformation process.

5 Conclusion and Future Work

The growing need for device independence and personalization forces WIS designers to support different kinds of adaptation. Another requirement towards such adaptive WISs is self-management, i.e. the ability to dynamically adjust themselves to a constantly changing environment. For that purpose this paper introduced the Self-Adapting GAC (SAG), a generic software component for adaptive data transformations with the ability of self-configuration. For its configuration an RDF-based rule language providing rules for content adaptation and self-configuration was introduced. The SAG's implementation was described in detail and a prototype application illustrating its main functions was developed.

Ongoing work concentrates on providing a more intuitive interface for developers aiming at configuring the SAG. This comprises both visual configuration tools as well as support for more high-level objects on top of XML constructs. Another task is the SAG's extension towards communication with heterogeneous data sources as well as multiple applications. Furthermore, we also to aim at using the developed RDF-based rule language in a broader context, especially within WIS specification frameworks.

References

[1] Isakowitz, T., Bieber, M., Vitali, F.: Web information systems - introduction. Communications of the ACM **41** (1998) 78–80

[2] Hori, M., Ono, K., Koyanagi, T., Abe, M.: Annotation by transformation for the automatic generation of content customization metadata. In: International Conference on Pervasive Computing, Pervasive 2002, Zurich, Switzerland. (2002)
[3] Passos, L.T., de Oliveira Valente, M.T.: Personalizing web sites for mobile devices using a graphical user interface. In: ICWE. (2004) 220–224
[4] Fiala, Z., Houben, G.J.: A generic trancoding tool for making web applications adaptive. In: The 17th Conference on Advanced Information Systems Engineering (CAiSE'05), Porto, Portugal. (2005)
[5] Fiala, Z., Hinz, M., Meiner, K., Wehner, F.: A component-based approach for adaptive dynamic web documents. Journal of Web Engineering, Rinton Press **2** (2003) 058–073
[6] Vdovjak, R., Frasincar, F., Houben, G.J., Barna, P.: Engineering semantic web information systems in Hera. Journal of Web Engineering, Rinton Press **2** (2003) 003–026
[7] Rossi, G., Schwabe, D., Guimaraes, R.: Designing personalized web applications. In: WWW10, The Tenth International Conference on the World Wide Web, Hong Kong. (2001)
[8] Lima, F., Schwabe, D.: Application modeling for the semantic web. In: First Latin American Web Congress (LA-WEB'03), IEEE (2003) 93–103
[9] Ceri, S., Fraternali, P., Bongio, A.: Web modeling language (WebML): a modeling language for designing web sites. In: 9th International Conference on the World Wide Web (WWW9), Amsterdam. (2000)
[10] Troyer, O.D., Casteleyn, S.: Designing localized web sites. In: WISE. (2004) 547–558
[11] Fiala, Z., Frasincar, F., Hinz, M., Houben, G.J., Barna, P., Meissner, K.: Engineering the presentation layer of adaptable web information systems. In: Fourth International Conference on Web Engineering (ICWE2004), Munich. (2004)
[12] Frasincar, F., Houben, G.J., Vdovjak, R.: Specification framework for engineering adaptive web applications. In: WWW11, The Eleventh International Conference on the World Wide Web. (2002)
[13] Fiala, Z.: Hera meets AMACONT - SAG prototype. http://www-mmt.inf.tu-dresden.de:8081/sag/index.html. (2005)
[14] Brusilovsky, P.: Adaptive hypermedia. User Modeling and User Adapted Interaction **11** (2001) 87–110
[15] Bra, P.D., Houben, G.J., Wu, H.: AHAM: A dexter-based reference model for adaptive hypermedia. In: 10th ACM Conference on Hypertext and Hypermedia (HYPERTEXT '99), Darmstadt, Germany, ACM (1999) 147–156
[16] Klyne, G., Reynolds, F., Woodrow, C., Ohto, H., Hjelm, J., Butler, M., Tran, L.: Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies. W3C Working Draft. (2003)
[17] Wireless Application Group, WAP Forum: User Agent Profile Specification. (2001)
[18] Hinz, M., Fiala, Z.: Amacont: A system architecture for adaptive multimedia web applications. In: Workshop XML Technologien fur das Semantic Web (XSW 2004). (2004)