A Generic Transcoding Tool for Making Web Applications Adaptive

Zoltán Fiala[1], Geert-Jan Houben[2]

[1] Technische Universität Dresden
Mommsenstr. 13, D-01062, Dresden, Germany
zoltan.fiala@inf.tu-dresden.de
[2] Technische Universiteit Eindhoven
PO Box 513, NL 5600 MB, Eindhoven, The Netherlands
houben@win.tue.nl

Abstract. As personalization and device independence become prominent issues in Web development, tools to adapt existing Web Information Systems (WISs) are required. Still, current development approaches lack support for adding adaptation to existing WISs. This paper presents the Generic Adaptation Component (GAC), a transcoding tool for making Web applications adaptable and adaptive. It can be seamlessly integrated into existing Web architectures as an autonomous adaptation and personalization module. For configuring the GAC an RDF-based rule language is presented, allowing to define both content adaptation and context data update rules. Moreover, based on the document generation architecture of the AMACONT project a collection of transformation operations is provided to realize these rules. The main functionality of the GAC is elucidated by an example implementation.

1 Introduction

The growing number of Web users with heterogeneous preferences and client devices makes personalization and device independence to central issues of Web development. Therefore, modern Web Information Systems (WIS) need to be extended with the ability to automatically adapt themselves to both individual users and their client platforms. However, recent WIS design frameworks that provide adaptation, e.g. [1, 2], assume to develop AWISs “from scratch” by using complex design models in which the adaptation is embedded. This leaves insufficient support for designers aiming at *adding adaptation* to existing WISs.

Recently, different transcoding solutions for adapting Web pages have been proposed. However, most of them are restricted to the presentation layer of Web applications, aiming at transforming HTML pages to limited device capabilities [3, 4] or users' visual impairments [5, 6]. Moreover, they typically support only static adaptation (*adaptability*), i.e. the adjustment of Web pages to a static set of user or device parameters. Still, we claim that the transcoding paradigm can be used for a broader range of adaptation and personalization issues, especially for *adaptivity*, i.e. adaptation according to parameters that may change while the Web presentation is being accessed or browsed.

Proceedings of the CAiSE'05 Forum - O. Belo, J. Eder, J. Falcão e Cunha, O. Pastor (Eds.)
© Faculdade de Engenharia da Universidade do Porto, Portugal 2005 - ISBN 972-752-078-2

For this reason, we introduce the *Generic Adaptation Component* (GAC) aiming at making existing Web applications adaptable and adaptive. The provider of a Web site can configure and integrate the GAC as a stand-alone module into the Web site architecture. We present an RDF-based rule language for specifying both *content adaptation* and *context data update rules*. Furthermore, based on the formats and the generation architecture of the AMACONT project [7], we provide a collection of operations that implement these rules.

2 GAC Architecture

Figure 1 shows how the GAC is integrated into the overall Web infrastructure. It processes XML-based *Web content* provided by some *Web application content generator* and adjusts it to the preferences and properties of individual users and their clients. The GAC performs different adaptations on its input, the recipe for which is specified by its *configuration*. This consists of a set of *adaptation rules*, each dictating a different content adaptation aspect. To take the current user and device context into account, adaptation rules can reference arbitrary parameters from the data describing the actual *adaptation context*. Finally, in order to support adaptivity, the configuration also contains *update rules* allowing to manipulate this context data according to the user's navigation and interaction history.

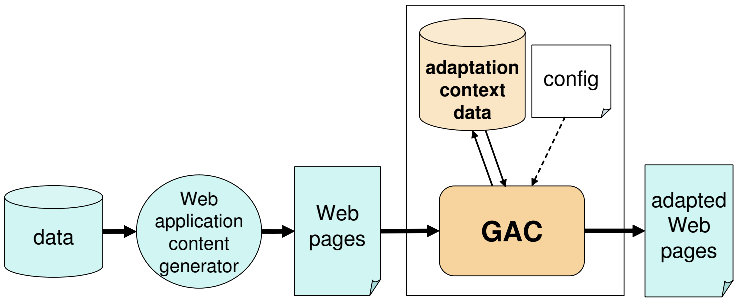

Fig. 1. System overview.

2.1 Input Data Requirements

For the sake of generality, arbitrary XML-based Web content is allowed as input for the GAC. This allows us to process a wide spectrum of Web content, both traditional Web pages delivered in a standardized presentation format ((X)HTML, cHTML or WML), as well as richly annotated XML data that abstracts from a specific output format and provides more information about the structure and semantics of its content. In general, the better structured and annotated the input data is, the more sophisticated adaptation rules can be specified.

In our running example [8] we use the GAC for adapting a dynamic WIS that provides information about the goals, products, and members of our research project called "Hera meets AMACONT!". As this WIS was developed following the model-driven Hera specification framework [2], the generated application is independent from a concrete implementation format and consists of a set of *slice* instances described in RDF. Abstracting from the notion of a Web page, a Hera slice is a meaningful grouping of content pieces that should be shown together in a hypermedia presentation. Since the delivered hypermedia presentation does not take into account the user's preferences, nor the client's capabilities, we use the GAC to add personalization and adaptation to it.

2.2 GAC Configuration

The GAC is controlled by its RDF-based configuration. It consists of a set of *rules* that specify the parts of the input data to be adjusted, the adaptations to be performed on them, and (in the case of adaptivity) the way the data defining the adaptation context has to be updated. A graphical excerpt of the RDFS schema defining our rule hierarchy is depicted in Figure 2. 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*.

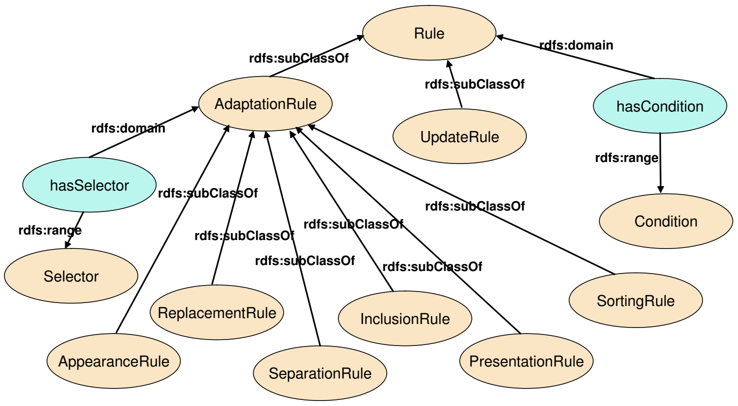

Fig. 2. GAC Rule Schema.

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 in different application scenarios, we identified a common set of generally applicable rule primitives. Based on Brusilovsky's survey on adaptive hypermedia [9], as well as our recent work on presentation adaptation [7], we selected and implemented the following:

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. Following code from our running example omits all slices containing images 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>
```

Separation Rules are a variation and less strict modification of *Appearance Rule*s. Instead of being elided, content units with invalid conditions are put on 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. A *Replacement Rule* substitutes XML fragments with alternative fragments.

Whereas the above rules address single content units, there are also rules adapting sets of content units, like 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 divide 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 personal 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. The @ID parameter used in the *by* attribute denotes the ID attribute of the currently selected XML element. Thus, the condition is appropriately adjusted to each selected project member (slice).

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

Note that this is an adaptivity example. The corresponding rule for updating the adaptation context data will be shown later.

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 [7], *Presentation Rules* assign so called *layout managers* to groups of data containers (slices in our example). Abstracting from the current browser's presentation capabilities, they dictate how the selected slices should be spatially arranged in the output presentation.

Update Rules aim at updating the adaptation context data. 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.

Since the above *Sorting Rule* requires to keep track of the pages (slices) already seen by the user, we show an Update Rule writing the IDs of visited slice instances to the application context data. The XPath expression identifies all slice instances by selecting elements whose name starts with the word "Slice".

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

3 Implementation

The GAC was realized with the aid of the AMACONT project's modular document generation architecture [7]. As illustrated in Figure 3, it consists of a configurable series of data transformation modules, each performing a specific content adaptation aspect. The document generator was realized based on the Cocoon publication framework.

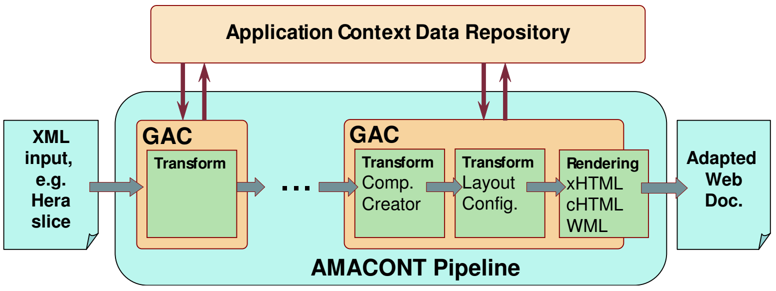

Fig. 3. Document Generation Pipeline.

Written in Java, the GAC was implemented as a custom Cocoon transformer. It communicates with the adaptation context data repository and performs the appropriate data transformations on the DOM view of its input documents. The repository was realized based on the open source RDF database Sesame. It stores context data relying on CC/PP, an RDF grammar for describing device capabilities and user preferences. The GAC utilizes SeRQL (Sesame RDF Query language) for retrieving or updating this data.

For illustrating the main functions provided by the GAC, a prototype application realizing the running example described in this paper was implemented [8].

4 Conclusion and Future Work

As personalization and device independence become key aspects of Web development, providers are often forced to add adaptation to an already available Web application. For that purpose we introduced the General Adaptation Component (GAC), an autonomous adaptation and personalization module. An RDF-based rule language was introduced for configuring the GAC by means of content adaptation and context data update rules. Based on the pipeline-based document generation architecture of AMACONT, we have implemented the GAC together with a collection of transformation operations as building blocks for the adaptation in a GAC configuration.

Future work aims at utilizing the GAC as a client-side component providing a private personalized view on Web applications for its users. As possible use cases we mention the adaptive management of bookmarks and links, the maintenance of users' personal comments (annotations) attached to Web page fragments, or even the "portable" realization of a WIS's adaptive presentation layer on a mobile device. Another targeted issue is the usage of the developed RDF-based rule language in a broader context, especially for the definition of complex adaptation and update rules within WIS specification frameworks.

References

[1] 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)

[2] 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

[3] Bickmore, T., Girgensohn, A., Sullivan, J.: Web page filtering and reauthoring for mobile users. Computer Journal **42(6)** (1999) 534–546

[4] Hori, M., Kondoh, G., Ono, K., Hirose, S.I., Singhal, S.: Annotation-based web content transcoding. In: 9th International Conference on the World Wide Web (WWW9), Amsterdam. (2000)

[5] Huang, A.W., Sundaresan, N.: A semantic transcoding system to adapt web services for users with disabilities. In: ASSETS'00, Arlington, Virginia. (2000)

[6] Yesilada, Y., Harper, S., Goble, C., Stevens, R.: Screen readers cannot see - ontology based semantic annotation for visually impaired web travellers. In: Fourth International Conference on Web Engineering (ICWE2004), Munich. (2004)

[7] 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

[8] Fiala, Z.: Hera meets AMACONT - GAC prototype. http://www-mmt.inf.tu-dresden.de:8081/gac/index.html. (2005)

[9] Brusilovsky, P.: Adaptive hypermedia. User Modeling and User Adapted Interaction **11** (2001) 87–110