Chapter 6

A Generic Transcoding Tool for Making Web Applications Adaptive

"This here's a re-search laboratory. Re-search means look again, don't it? Means they're looking for something they found once and it got away somehow, and now they got to re-search for it?"[1]

6.1 Motivation and Introduction

In the preceding chapters of this thesis a component-based document format for adaptive dynamic Web documents was introduced. In combination with a structured authoring process and supported by a graphical authoring tool, it facilitates the efficient development of personalized ubiquitous Web presentations from reusable implementation artefacts. It was illustrated how different application aspects (concerning content, navigation, presentation, and their appropriate adaptation issues) can be systematically considered by guiding component developers through the phases of the overall Web engineering process. However, the resulting authoring framework assumes to create adaptive Web applications "from scratch", not providing sufficient support for developers (providers) who intend to *add adaptation* to an already existing Web-based system.

On the other hand, there already exists a number of formats, methodologies, and frameworks for Web application engineering. A detailed overview of the most important approaches was provided in Chapter 3. As discussed there, only some of them support (selected) issues of personalization and device dependency. Therefore, this chapter deals with the question how the lessons learned from engineering component-based adaptive Web presentations can be applied (i.e. generalized) for extending a broader range of existing Web applications by additional adaptation concerns.

In order to answer this question, it is important to investigate the way how existing Web Information Systems are typically implemented. As can be observed, they are generally based on a series of data transformations that convert some input content (in general XML data) to a hypermedia presentation in a particular implementation format, such as (X)HTML, cHTML, WML, X3D, etc. These data transformations are controlled by a specification (mostly in form of a specific XML-based document format) that dictates the application's semantic, navigational, and presentational behavior [Fiala and Houben 2005]. Such a typical pipeline-based (staged) Web presentation generation architecture is illustrated in Figure 6.1. The original content is subject to a number of transformations that subsequently lead to the desired hypermedia presentation.

Note that Web Information Systems supporting adaptation are realized in a similar way.

[1]Kurt Vonnegut, Jr.: Cat's Cradle, 1963

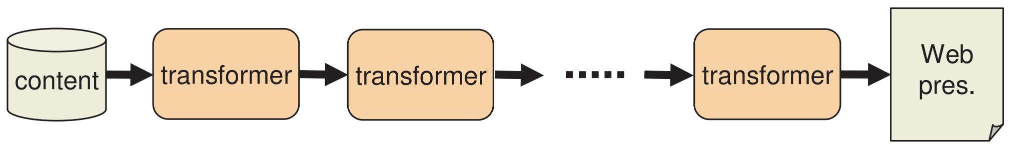


Figure 6.1: WIS implementation based on data transformations

The only difference to non-adaptive WISs is that they utilize adaptation-specific content transformation steps that are additionally parameterized by available external information describing the user's actual usage context. Usually, this information is referred to as a *user model* or a *context model*. It is typically used (referenced) by *adaptation conditions* that are attached to (i.e. contained by) fragments of the input content. A content transformation pipeline containing such an adaptation-specific transformation step is illustrated in Figure 6.2. Note, however, that there are also scenarios where several transformation steps utilize context information.

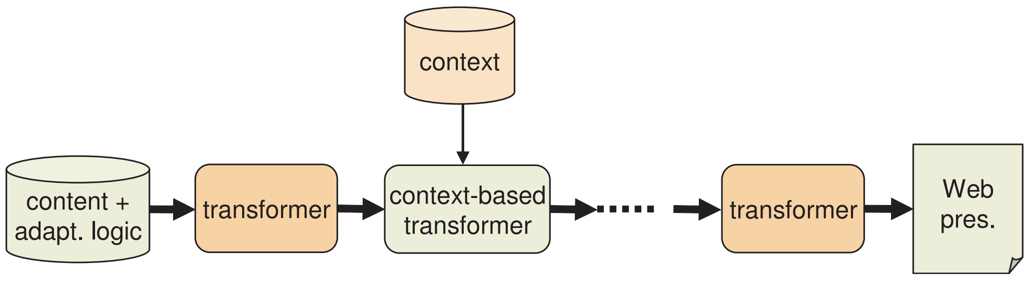


Figure 6.2: WIS implementation with adaptation

In general, most content transformations in a WIS implementation are very specific to the formats or models dictated by the underlying methodology or framework. Nevertheless, it can be recognized that adaptation-specific transformations have a lot in common. As identified by Brusilovsky's surveys [Brusilovsky 2001], they typically perform similar operations on (groups of) structured content fragments (e.g. document components, data units, slices, document nodes). Well-known adaptation operations on such structured content fragments are *conditional inclusion*, *selection*, *removal*, *insertion*, *sorting*, etc. (see Section 2.2.3). As also demonstrated by the component-based document format introduced in Chapter 4, such basic adaptation operations (transformations) can be used to realize a variety of adaptation concerns.

Given the similarity and the generic nature of such “adaptation-specific content transformations”, the key observation can be made that major parts of them can be well separated from the rest of a Web application's hypermedia generation pipeline. What is more, this separate implementation of selected adaptation transformations (operations) also allows for “extracting” their configuration from the document formats describing the underlying Web application. As a consequence, it becomes possible to realize given adaptations based on *generic* transformer modules that can be appropriately controlled by an *external* configuration. Moreover, when both the implementation and appropriate configuration of adaptation operations can be separated from the original application, then it also becomes possible to *add adaptation* to an existing Web-based system.

A transformation scenario utilizing such a generic transformer module is depicted in Figure 6.3. Note that besides the information describing the current usage *context*, this transformer additionally takes an external *adaptation recipe* (i.e. configuration) into account, that dictates which adaptation operations it has to perform on (which selected parts of) its input content. That is to say, the specification of adaptations is not an inherent part of the input content anymore. Quite the opposite, it is "outsourced" to the generic transformer's configuration and addresses the content fragments (e.g. document components, data units, sections, etc.) to be adapted externally. Thus, the concept of document components (fragments) containing inherent adaptation descriptions can be generalized for a broader range of Web applications by the external assignment of adaptation descriptions to parts (fragments) of an arbitrary XML-based document format.

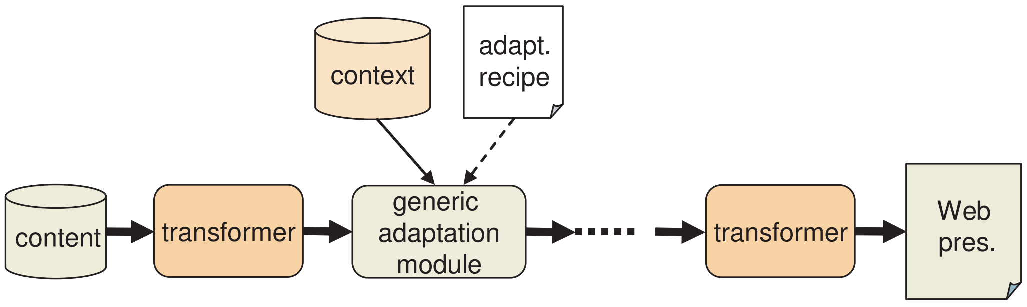


Figure 6.3: WIS implementation based on generic adaptation modules

To demonstrate this idea, this chapter introduces the Generic Adaptation Component (GAC [Fiala and Houben 2005]), a generic transcoding tool aiming at making existing Web applications adaptable and adaptive[2]. The provider of a Web Information System can use it as a stand-alone module, configure it, and integrate it into his Web site architecture. For the configuration of the GAC an RDF-based rule language is introduced, allowing to specify rules for both content adaptation and context data updates. Furthermore, a set of operations for implementing these rules will be provided.

Note that the separation of adaptation from the rest of the application might obviously result in a restricted adaptation support compared to adaptive Web applications that have been designed for adaptation from the beginning. However, it will be demonstrated that even this transformation-based "lightweight" adaptation extension can be efficiently utilized in many different application scenarios.

The rest of this chapter is structured as follows. After briefly discussing related approaches in Section 6.2, an overview of the architecture, the main functionality, and the most important application scenarios of the GAC is given (Section 6.3). Then, Section 6.4 describes central issues of the GAC's configuration in more detail, among them the requirements towards its input data, the adaptation context data it utilizes, and its RDF-based configuration language. To help the reader understand the main concepts, all these aspects are explained by a running example. The implementation details of the GAC based on the component-based document format's presentation generation architecture are introduced in Section 6.5. Finally, Section 6.6 gives a comparison of the transcoding-based adaptation approach described in this chapter and the component-based approach explained in the previous chapters, by discussing their main advantages and disadvantages.

[2]The GAC was designed and developed within the scope of a long-term collaboration between the author's research group (the AMACONT project [@AMACONT]) and the Hera research program [@HERA]. Note, however, that the GAC's basic concepts, its rule-based configuration language, and its implementation are a contribution of the author.

6.2 Existing Web Transcoding Solutions

Recently, a number of transcoding solutions for adapting Web applications have emerged. Most of them aim at adjusting HTML-based Web content to limited presentation capabilities, like those of small hand-held devices [Alam and Rahman 2003].

Many approaches utilize so-called transcoding heuristics [Bickmore et al. 1999] for the fully automatic re-authoring of Web pages. The most important characteristics of such heuristics is that they do not take into account the structure or semantics of a particular Web page, thus they provide transcoding operations that are applicable to (almost) arbitrary Web pages. The most widely used heuristics are first sentence elision, image reduction [Bickmore et al. 1999], video/audio transcoding [Smith et al. 1998], and outlining (i.e. the replacement of section headers with hyperlinks pointing to the corresponding text blocks [Hwang et al. 2002]). Furthermore, there are also approaches aimed at removing advertisements, link lists, or empty tables [Gupta et al. 2003], as well as automatically abbreviating common words [Gomes et al. 2001]. While being generally applicable, these approaches do not take into account the specific structure and the domain-specific semantics of the underlying Web content sufficiently. Furthermore, as a consequence of their heuristic nature, the result of the adaptation (and especially the quality or usability of the resulting Web pages) is often unpredictable [Hwang et al. 2003].

Barrett et al. [Barrett and Maglio 1999] define intermediaries as computational entities that operate on information as it flows along a stream, and introduce the Web Intermediaries (WBI) [Barrett et al. 1997] approach, a framework for manipulating Web information streams. Data manipulation functionality is implemented by autonomous agents that can be deployed on the server, the client, or as proxies. The approach supports four kinds of agents: monitors, editors, generators, and autonomous agents. Utilizing WBI, Hori et al. [Hori et al. 2000] present an annotation-based transcoding solution for accessing HTML documents from information appliances like PDAs, cell phones, and set-top boxes. RDF-based external annotations specifying content transformation rules such as *content alternative selection* or *page splitting hints* can be assigned to fragments of particular Web pages [Hori et al. 2002]. The main benefit of this approach is that both the structure and content of the input data can be taken into account. Furthermore, as the adaptation metadata is separated from the content itself, different application-specific adjustment scenarios are possible. However, even this approach is restricted to the transcoding of HTML content mainly based on device capabilities. There is no support for dynamic adaptation, nor for maintaining a broader range of contexts (e.g. personalized user profiles).

A similar solution based on the assignment of external transformation instructions to Web documents is provided by RDL/TT (Rule Description Language for Tree Transformations [Schaefer et al. 2002, Osterdiekhoff 2004]). Still, instead of declarative annotations, a Java-based imperative transcoding language is utilized. Again, this language focuses also primarily on the specifics of HTML-based Web documents.

Besides for device adaptation, transcoding techniques are also intensively used to make Web applications accessible for visually impaired users [Asakawa and Takagi 2000]. Again, some solutions are based on external annotations. As an example, we mention the Travel Ontology [Yesilada et al. 2004], allowing to (semi-)automatically transform Web pages to a form optimized for voice output. Aurora [Huang and Sundaresan 2000] pursues a more semantic approach and uses domain-specific schemas describing the functional semantics of Web objects to extract their content and automatically adapt it to different user requests.

Looking at related work on Web transcoding, one can see that existing approaches mainly allow static adaptation (*adaptability*), i.e. the adjustment of Web pages to a static set of user or device parameters. Moreover, most solutions are restricted to the presentation layer of Web applications, aiming at transforming HTML pages to limited device capabilities or users' visual impairments. Still, we claim that transcoding could 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.

6.3 GAC: Generic Adaptation Component

6.3.1 GAC Overview

As mentioned above, the GAC is a generic transcoding tool aimed at adding adaptation to existing Web applications. Figure 6.4 shows how it is integrated into a typical hypermedia generation process: it processes XML-based *Web content* provided by some *Web application generator* and adjusts it to the preferences and properties of individual users and their clients. As a generic component, 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 (besides the input) the current usage context into account, adaptation rules can reference arbitrary parameters from the *adaptation context data*. 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.

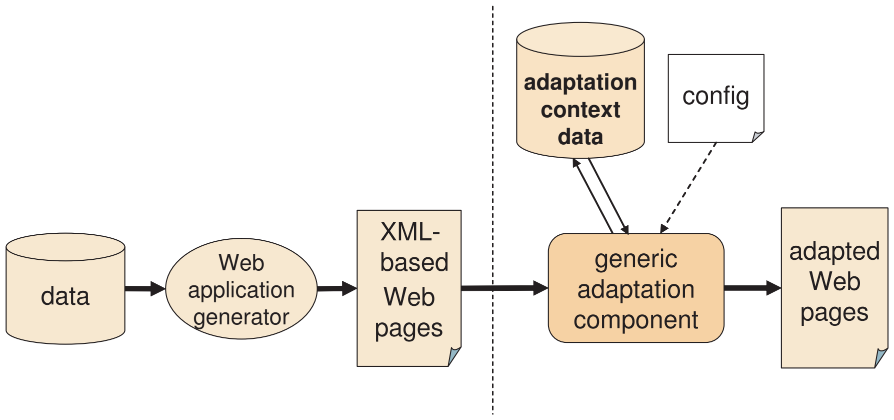


Figure 6.4: GAC abstract system overview

6.3.2 Possible Application Scenarios

As a generically applicable transcoding tool, the GAC does not make any assumptions (restrictions) to the preceding Web content generation process[3]. Quite the opposite, it can support a variety of application scenarios depending on how its XML-based input is created or generated. The following examples summarize a number of important GAC application areas.

[3]As will be described in more detail in Section 6.4.1, the only assumption is that the the original content generation process delivers data in XML format.

Transcoding static Web pages

Figure 6.5 shows a basic transcoding scenario where the "Web application generation process" acts as a traditional Web server delivering static XHTML pages. As an example, the GAC could adapt those pages to limited devices by filtering out large images, omitting videos, or eliding tags not interpretable on them. Furthermore, it could also perform user specific personalization tasks, such as changing font colors or removing information being unimportant for the user (e.g. decoration elements or links to forbidden sites).

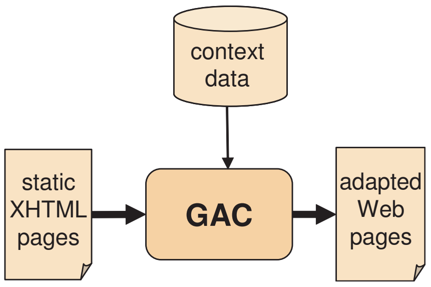


Figure 6.5: GAC scenario 1. - Transcoding static XHTML

Adaptive WIS Front-end

A more complex scenario is shown in Figure 6.6. In this case the GAC is used as the adaptive front-end of a more complex Web Information System. Based on its input data (which is typically retrieved from a data source, e.g. a database), this WIS delivers dynamically generated Web pages.

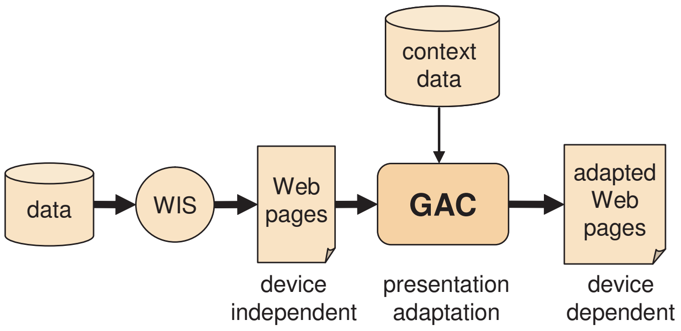


Figure 6.6: GAC scenario 2. - Adaptive WIS front-end

The WIS might be non-adaptive, or it might have already performed some content- or navigation-specific adaptations. Consequently, the role of the GAC could be to perform presentation-specific adaptation operations on those pages. This could include the adjustment of content elements to the media types and tag structures supported by a specific device or document format, the manipulation of the spatial adjustment of those content elements on the generated pages, or even the consideration of the current user's layout preferences (background images, link colors, etc.). However, the fact that the GAC operates "only" on the presentation level of the underlying Web application means that its adaptation capabilities are limited, respectively.

Transcoding with multiple GACs

While the above examples utilize only one GAC, it is possible to employ several independent GACs at different stages of the Web presentation generation process. As discussed in Section 5.3.2, a Web Information System can be efficiently realized with three layers, namely the semantic layer, the navigation layer, and the presentation layer, each responsible for its specific adaptation processes. Thus, as an extension of the preceding scenario, a non-adaptive WIS can be extended with a GAC each layer (see Figure 6.7). As a matter of course, each GAC is required to deliver data corresponding to the requirements of its successor layer.

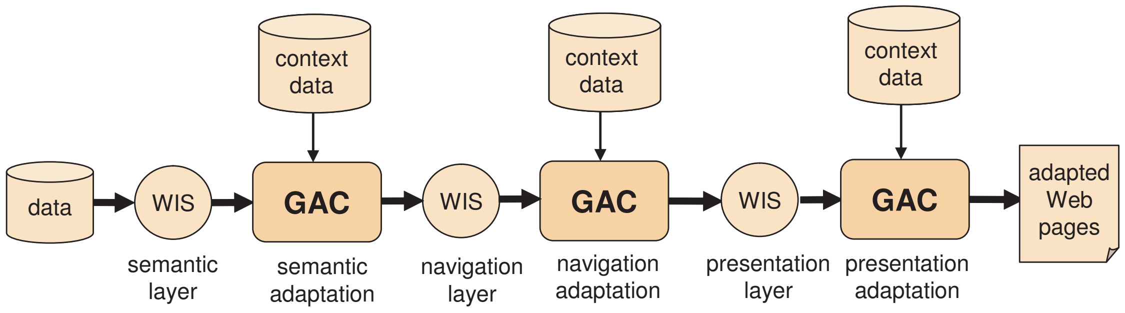


Figure 6.7: GAC scenario 3. - Adaptive WIS based on GAC pipeline

Separation of Adaptation Aspects with Multiple GACs

The GAC is a generic component aimed at performing different kinds of adaptations on its input data. Still, adaptation in a Web application is typically centered around a number of well separable independent adaptation concerns (or adaptation aspects). For instance, the navigational structure of a Web application might be adjusted according to a number of (possibly orthogonal) design concerns, such as device dependency, localization, personalization, or security. While all of these adaptations can be reduced to context-based data transformations, the provider of a Web application might need to handle them independently, i.e. by using a separate GAC for each of them.

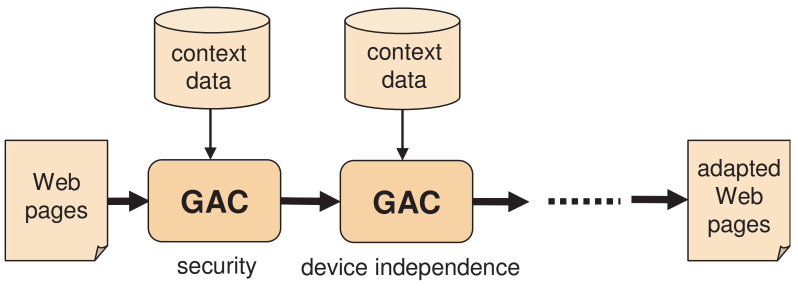


Figure 6.8: GAC scenario 4. - Separation of concerns with multiple GACs

Figure 6.8 illustrates such an adaptation scenario consisting of two GACs. While the first one performs adaptation operations supporting security issues (e.g. by hiding trustworthy content from users that are logged in as guests), the second one targets device independence (e.g. by filtering out media items being not suitable for a certain client device). Note that this separation of adaptation aspects allows providers to easily add (or remove) additional adaptation concerns to an application without the need to change (reconfigure) or rewrite it completely. Furthermore, they can also easily reconfigure the priority of adaptation concerns by exchanging the order according to which the utilized GACs are switched in line. For more information on the advantages of this separation of concerns by using multiple GACs the reader is referred to [Casteleyn et al. 2006a, Casteleyn et al. 2006b].

Adaptivity Support

As mentioned in Section 6.2, most transcoding-based solutions provide adaptability, i.e. the adaptation based on a static user or device profile, not taking into account the user's browsing behavior. Still, modern AWISs with their increased interactivity require to support dynamic adaptation (adaptivity). As examples we mention the elision of information the (returning) user has already seen, the recommendation of links to pages the user might have become interested in, but also the dynamic reorganization of the WIS's presentation layer whenever he resizes his browser window.

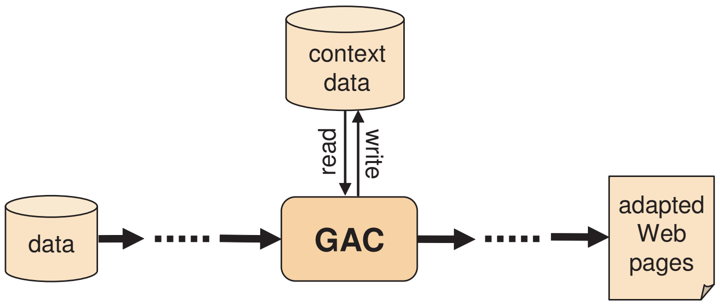


Figure 6.9: GAC scenario 5. - Support for adaptivity

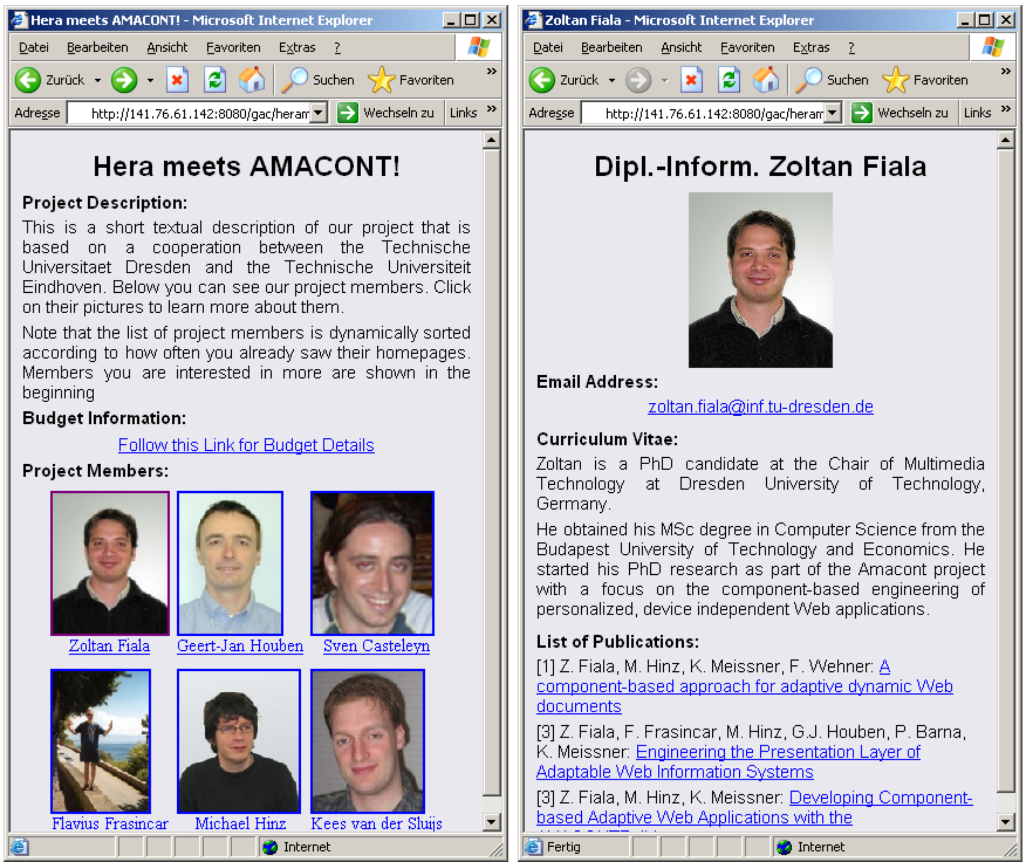
To support adaptivity, the GAC has access to adaptation context data, and it can not only read but also dynamically update that context data by means of so-called *update rules* (see Figure 6.9). Consequently, the example scenarios mentioned above can be extended by even more sophisticated adaptation mechanisms. As a possible extension of the first scenario, the GAC can monitor the pages visited by users, maintain the knowledge they obtain when reading those pages, and use this knowledge to dynamically order links to related pages.

6.3.3 Running Example Overview

In order to help the reader understand the main concepts, the architecture, and the configuration of the GAC, the rest of this chapter will explain these details based on a small example application. This selected example is a dynamic Web Information System providing information about a research collaboration between the author's research group (the AMACONT project [@AMACONT]) and the Hera research program of the Vrije Universiteit Brussels [@HERA]. There are members working at the project, each characterized by a name, a CV, a picture, as well as some contact data (email address, phone number etc). They produce publications on their research efforts, which are described by a title, the name of the corresponding conference or journal, the year of publication, and an abstract.

The example Web application consists of dynamically generated Web pages in the XHTML format (see Figure 6.10). The starting page of the example application is the project homepage. It provides basic information on the project (title, description, budget information) as well as a dynamically generated link list consisting of its members' pictures as thumbnails.

By clicking on a thumbnail the user can navigate to the corresponding member's page that contains his name, contacts, CV, image, and a list of his publications. This list again leads to another page describing the corresponding publication in more detail.


Hera meets AMACONT! - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten
Adresse http://141.76.61.142:8080/gac/heram Wechseln zu Links
Internet

Hera meets AMACONT!

Project Description:

This is a short textual description of our project that is based on a cooperation between the Technische Universitaet Dresden and the Technische Universiteit Eindhoven. Below you can see our project members. Click on their pictures to learn more about them.

Note that the list of project members is dynamically sorted according to how often you already saw their homepages. Members you are interested in more are shown in the beginning

Budget Information:

Follow this Link for Budget Details

Project Members:

Zoltan Fiala, Geert-Jan Houben, Sven Casteleyn, Flavius Frasincar, Michael Hinz, Kees van der Sluijs

Zoltan Fiala - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten
Adresse http://141.76.61.142:8080/gac/heram Wechseln zu Links
Fertig
Internet

Dipl.-Inform. Zoltan Fiala

Email Address:

zoltan.fiala@inf.tu-dresden.de

Curriculum Vitae:

Zoltan is a PhD candidate at the Chair of Multimedia Technology at Dresden University of Technology, Germany.

He obtained his MSc degree in Computer Science from the Budapest University of Technology and Economics. He started his PhD research as part of the Amacont project with a focus on the component-based engineering of personalized, device independent Web applications.

List of Publications:

[1] Z. Fiala, M. Hinz, K. Meissner, F. Wehner: A component-based approach for adaptive dynamic Web documents

[3] Z. Fiala, F. Frasincar, M. Hinz, G.J. Houben, P. Barna, K. Meissner: Engineering the Presentation Layer of Adaptable Web Information Systems

[3] Z. Fiala, M. Hinz, K. Meissner: Developing Component-based Adaptive Web Applications with the

Figure 6.10: GAC running example overview

As this Web presentation does not take into account the user's preferences, nor the client's capabilities, the GAC will be used to add personalization and adaptation to it. The configuration and realization of the supported adaptations will be shown in the following.

6.4 GAC Configuration

As shown in Figure 6.4, the most crucial issues for understanding the overall architecture and functionality of the GAC are 1) the requirements towards the input content to be adapted, 2) the structure of the adaptation context data, and 3) the RDF-based rule language used to configure the corresponding adaptation operations. In accordance with the running example described above, this section explaines these issues in more detail.

6.4.1 Input Data Requirements

The GAC gets its input data from a Web presentation generation process, which can be e.g. a (part of a) legacy Web application. According to its configuration and the information describing the current usage context, it performs transformations on that input. Thus, the transformations need access to the input content, i.e. a definition (of an interface) is required that states the structural elements to be encountered in it.

For the sake of generality, arbitrary XML-based Web content is allowed as input for the GAC. This enables the GAC to process a wide spectrum of content, both Web pages delivered in a standardized 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 adaptations can be specified.

As discussed above, the example application (to be adapted) used throughout this chapter delivers Web pages in XHTML. Furthermore, it is assumed that its designer put a focus on the separation of content and layout, and structured the generated presentation appropriately. Listing 6.1 shows the structure of a Web page presenting information about a project member.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" ... >
<html>
 ...
 <body>
  <div class="member" id="member_ID1">
   ....

   <div class="membername" id="membername_ID1">
    <h1>Dipl-Inform. Zoltán Fiala</h1>
   </div>
   ...

   <div class="membercv" id="membercv_ID1">
    <p>Zoltán is a PhD student at Dresden University of Technology ...</p>
   </div>
   ...

   <div class="publications" id="publications_ID1">
    ....
   </div>

  </div>
 ...
 </body>
</html>
```

Listing 6.1: GAC input content example

As can be seen, the meaningful content elements (e.g. the member's name, CV, publications, etc.) to be presented are appropriately encapsulated by *div* elements and are also identified by a *class* attribute and a unique *id* attribute. As will be shown later, the presence of such content structuring tags facilitates to apply a number of content-specific adaptations. Note, however, that the usage of this structure in our example does not restrict the GAC's generality. First, it will be shown that GAC rules are independent of specific XML grammars. Second, most hypermedia document formats and WIS approaches utilize similar hierarchically ordered data containers to structure their Web content. As important analogies we mention WebML's data units [Ceri et al. 2000], Hera's slices [Frasincar et al. 2002], AHA!'s page fragments [De Bra et al. 2002], CHAMELEON's components [Wehner and Lorz 2001], HMDoc's document nodes [Westbomke and Dittrich 2002], or even the Section elements of the upcoming W3C standard XHTML 2.0 [Axelsson et al. 2004].

6.4.2 Adaptation Context Data

The adaptation operations executed by the GAC are parameterized by the *adaptation context data*. It contains up-to-date information on the user, his device, and entire usage context which the GAC has read and write access to. The structure of the GAC's adaptation context data is based on CC/PP, an RDF grammar for describing device capabilities and user preferences. As mentioned in Section 4.5.2, it represents context information based on a two-level hierarchy of *components* and their *attributes*, the latter of which are described as name-value pairs.

The GAC's configuration language (which will be described in more detail in Section 6.4.3) refers to context data parameters as variables of the form `$paramname`, where `paramname` is a literal consisting of alphanumeric characters. Moreover, it also allows for array-like context parameters in the form `$paramname[index]`, where the index of such an array is again an arbitrary literal of alphanumeric characters[4]. The usage of such array-like structures is important when handling context information which is somehow related to the underlying data (e.g. the number of times the user was been presented a given content element) and will be demonstrated in Section 6.4.3 by a number of examples.

As it will be shown later, the usage of CC/PP allows to "reuse" the context modeling framework of the modular document generation architecture presented in Section 4.5 for the GAC's implementation. An excerpt from the CC/PP-based context model of that architecture was already shown in Section 4.5.2. As mentioned there, it can be extended arbitrarily by the introduction of new profiles.

6.4.3 The Rule-based GAC Configuration Language

The GAC is controlled by its RDF-based configuration. It consists of a set of *rules* that specify 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. Rules are declarative, i.e. they describe *what* should be done, rather than *how*. This means, for example, that different implementations are possible for a rule specification. This is a main benefit compared to imperative approaches (e.g. RDL/TT [Schaefer et al. 2002]) that explicitly focus on a concrete implementation.

A graphical excerpt of (a part of) the RDF schema defining the GAC rule hierarchy is depicted in Figure 6.11. 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 consisting of constants, parameters from the adaptation context data, as well as logical and arithmetic operations. Rules can be either *adaptation rules* or *update rules*. Whereas *adaptation rules* describe how the input data has to be transcoded, *update rules* aim at manipulating the adaptation context data. Along the lines of the example application, the following sections describe the corresponding rule types and their configuration options in more detail.

6.4.4 Adaptation Rules

Adaptation rules describe basic adaptation operations to be performed on specific parts or structures of the input content. As depicted in Figure 6.11, they all inherit from the abstract class *AdaptationRule*. They have a *selector* property that contains an XPath expression

[4]In order to stay conform with the RDF-based syntax of CC/PP, such parameters are serialized as RDF properties called "paramname_index".

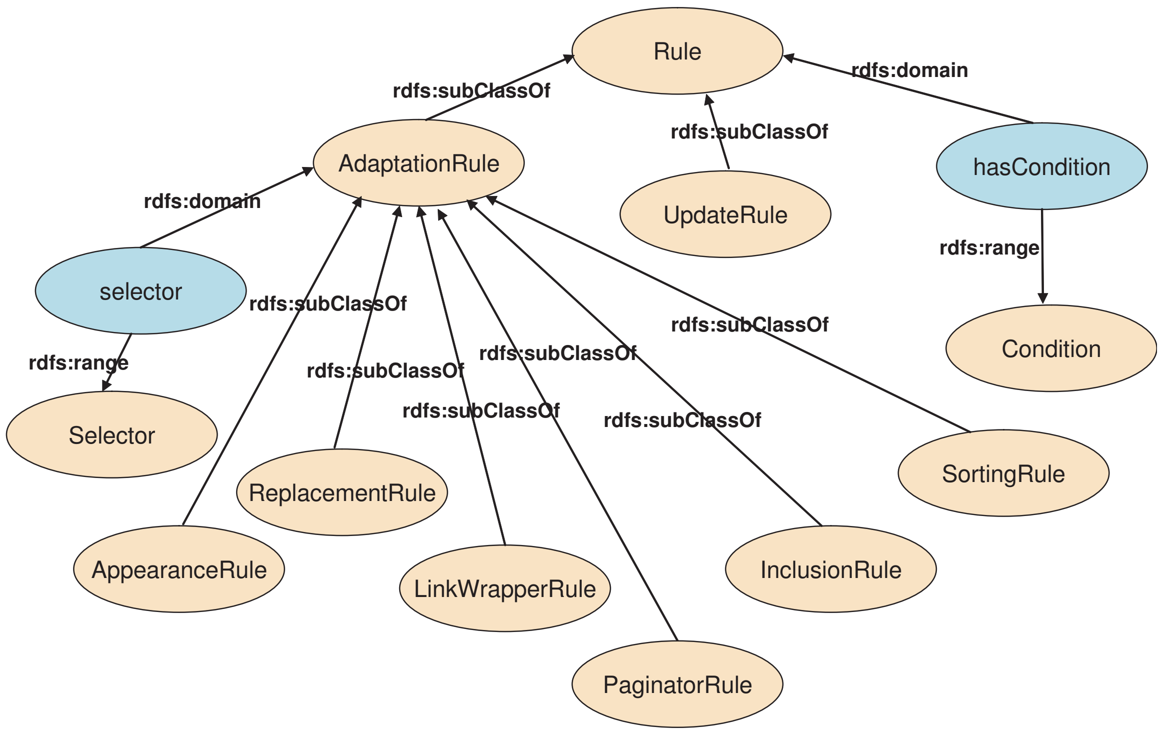


Figure 6.11: GAC rule schema excerpt

[Berglund et al. 2004] in order to unequivocally identify the parts of the XML input content to be adapted. Whenever there are several *adaptation rules* addressing the same part of the input content, they are ordered according their *priority* properties. The *priority* property of an *adaptation rule* is a non negative integer value. Its usage is optional, the default priority value is 0. *Adaptation rules* of a given GAC configuration are executed according to the descending order of their priorities, i.e. the rule with the highest priority property is processed first. *Adaptation rules* with the same priority are executed according to the order of their occurrence in the GAC's RDF-based configuration document.

Table 6.1 summarizes the properties used for parameterizing *adaptation rules*. It specifies their names, their meaning, their usage (i.e. whether they are required or optional), as well as their possible values.

Name	**Meaning**	**Usage**	**Values**
`selector`	Identifies the parts of the input content to be transcoded	required	XPath expression
`priority`	Rule priority	optional	Non negative integer

Table 6.1: Properties of an adaptation rule

In order to use *adaptation rules* (and the corresponding adaptation operations) in different application scenarios, a common set of generally applicable rule primitives were identified. Based on Brusilovsky's survey on basic methods and techniques for content and navigation adaptation [Brusilovsky 2001], as well as own results concerning presentation adaptation [Fiala et al. 2004a], following rules have been selected and implemented[5]:

6.4.4.1 Appearance Rule

An *appearance rule* (Class *AppearanceRule*) realizes one of the most basic adaptation methods: the selected content is included in the output only if the associated condition is valid. Appearance rules can address arbitrary (sets of) XML elements, attributes, and text nodes. In the case when an element is selected, all of its descendant nodes are concerned, as well. An appearance rule has no additional parameters.

The two rule examples in Listing 6.2 address device dependency by adjusting the original Web content to the limited presentation capabilities of handheld devices (PDAs). While the first one omits all images from the input XHTML documents, the second one removes the lists of publications from them. The XML elements (fragments) to be elided are selected by the XPath expressions in the rules' *selector* properties (see lines 1 and 8). The conditions address parameters from the adaptation context data which are denoted with a $ sign, respectively.

```
1     <gac:AppearanceRule rdf:ID="hideImageRule" gac:selector="//img">
2       <gac:hasCondition>
3         <gac:Condition gac:when="($Device!='pda')"/>
4       </gac:hasCondition>
5     </gac:AppearanceRule>
6
7     <gac:AppearanceRule rdf:ID="hidePubListRule"
8                         gac:selector="//div[@class=`publications`]">
9       <gac:hasCondition>
10        <gac:Condition gac:when="($Device!='pda')"/>
11      </gac:hasCondition>
12    </gac:AppearanceRule>
```

Listing 6.2: Appearance rule example

While these appearance rules realize static adaptation, Listing 6.3 also defines one for adaptivity. The CV of a project member is shown only for users who visit that page for the first time (i.e. it is elided for returning users). The `Visited` variable from the adaptation context data is in this case an array which is parameterized by the `id` attribute of the currently selected XML element (in this case the element representing members' CVs). This way the condition is appropriately adjusted to each selected CV instance. As a matter of course, this example assumes that the user's visits to project members' homepages (and CVs) are appropriately tracked during his navigation through the Web application. The corresponding rule for updating the adaptation context data will be shown later in Section 6.4.5.

6.4.4.2 Element Filter Rule

While an appearance rule allows to include/exclude a whole XML fragment, in some cases it is meaningful to filter out only an element itself and retain its descendant elements (nodes). This is typically the case when processing XML elements aimed at formatting the underlying content, such as the *b* (bold), *i* (italics) or *u* (underline) tags of HTML. For this purpose the so-called *element filter rule* (Class *ElementFilterRule*) was developed. Its selector property

[5]Since these rules inherit from the abstract AdaptationRule, only their additional properties will be mentioned and specified in the rest of this chapter.

```
<gac:AppearanceRule rdf:ID="hideCVRule"
                    gac:selector="//div[@class='membercv']">
  <gac:hasCondition>
    <gac:Condition gac:when="($Visited[@id]==0)"/>
  </gac:hasCondition>
</gac:AppearanceRule>
```

Listing 6.3: Appearance rule example Nr. 2

can address arbitrary XML elements in the input content. Similar to appearance rules, it has no additional parameters.

The example shown in Listing 6.4 disables all hyperlinks pointing to the detailed description of publications (on the pages of project members) for users that are logged in as guests. It filters out all corresponding *a* elements but does not remove the link anchors (in this case the titles of the publications). Thus, in this particular scenario this rule implements the adaptive navigation technique *link disabling* (see Section 2.2.3).

```
<gac:ElementFilterRule rdf:ID="disablePubLinksRule"
                    gac:selector="//div[@class='publications']//a">
  <gac:hasCondition>
    <gac:Condition gac:when="$Login=='Guest'"/>
  </gac:hasCondition>
</gac:ElementFilterRule>
```

Listing 6.4: Element filter rule example

6.4.4.3 Inclusion Rules

An *inclusion rule* (Class *InclusionRule*) realizes the inverse mechanism, i.e. the insertion of external content into the processed Web document. If the associated condition is valid, XML data from a specific URI (specified by the additional *what* property of the *inclusion rule*) is included in the output document at the place determined by the rule's selector (see Table 6.2).

The data to be inserted has to be well-formed XML. Furthermore, the selector property of an inclusion rule has to address an element node. The optional *where* property defines whether the data to be included should be inserted as a preceding sibling, a following sibling, or as the first child element of the selected XML element. Its default value is *child*. Whenever the values *preceding* or *following* are used, the selected XML element may not be the input XML document's root element. Note that this generic mechanism is a powerful means of data insertion: the addressed URI can be e.g. the target URL of an HTTP request or even a complex query to a dynamic data source.

The example shown in Listing 6.5 includes an advertisement at the bottom of the project homepage for desktop PCs.

While the inclusion rule facilitates the insertion of an arbitrary XML fragment, the *attribute inclusion rule* (class AttributeInclusionRule) aims at inserting an XML attribute into an XML element. It has two additional properties that indicate the new attribute's name and value, respectively (see Table 6.3).

Name	Meaning	Usage	Values
`what`	URL pointing to the content to be included	required	String describing an URI
`where`	relative position to the selected element	optional	preceding|following|child

Table 6.2: Properties of an inclusion rule

```
<gac:InclusionRule rdf:ID="includeAdvertRule"
                   gac:selector="//div[@class='project']/*[last()]">
  <gac:hasCondition>
    <gac:Condition gac:when="($Device=='desktop')"/>
  </gac:hasCondition>
 <gac:what>http://www-mmt.inf.tu-dresden.de/gacadvert</gac:what>
 <gac:where>preceding</gac:where>
</gac:InclusionRule>
```

Listing 6.5: Inclusion rule example

6.4.4.4 Replacement Rules

A *replacement rule* (Class *ReplacementRule*) substitutes specific parts of the input content with an alternative value. Its selector property can address XML elements, attributes, or text nodes. The additional *with* parameter specifies the new value of the selected document part and is a string (optionally containing context data parameters). While in the case of XML elements their names are changed, attributes and text nodes get a new value.

As an example, the simple replacement rule shown in Listing 6.6 enlarges XHTML headers for users with visual impairments by exchanging H3 tags with H1 elements.

```
<gac:ReplacementRule rdf:ID="replaceHeaderRule"
                     gac:selector="//H3">
  <gac:hasCondition>
    <gac:Condition gac:when="($visuallyimpaired=='yes')"/>
  </gac:hasCondition>
  <gac:with>H1</gac:with>
</gac:ReplacementRule>
```

Listing 6.6: Replacement rule example

While the replacement rule allows the manipulation of single XML tags, the *code replace-*

Name	Meaning	Usage	Values
`name`	name of the attribute to be included	required	String
`value`	value of the attribute to be inserted	required	String

Table 6.3: Properties of an attribute inclusion rule

Name	**Meaning**	**Usage**	**Values**
`with`	Indicates a new element name, attribute value or text node content	required	String

Table 6.4: Properties of a replacement rule

ment rule (Class *CodeReplacementRule*) is a slightly modified version of it that enables to replace larger XML code fragments. The rule's selector property addresses the starting XML element of the document fragment to be replaced. However, in this case the *with* property is not a simple literal, rather a URI addressing a "remote" code fragment (see Table 6.5).

Name	**Meaning**	**Usage**	**Values**
`with`	URI pointing to an XML fragment	required	String

Table 6.5: Properties of a code replacement rule

6.4.4.5 Link Wrapper Rule

Link wrapper rules (Class *LinkWrapperRule*) are used to manipulate the target URLs of hyperlinks found in the input content. Their main application is the modification of hyperlink targets encountered in the input XML documents. In this way users' clicks on the appropriate links can be redirected to a target defined by the GAC configurator. Furthermore, according to the adaptation context data the link wrapper rules can also add additional (personalized) request parameters to hyperlinks.

In order to identify the hyperlink references (URLs) to be manipulated the rule's selector property is used. The *toURL* property specifies the new URL (to which the link has to be redirected). Furthermore, a number of parameters in form of name/value pairs can be defined in order to attach arbitrary request parameters to the link (see Table 6.6). The optional *keepOldURL* parameter indicates that the original URL should be retained as a special request parameter of the new URL. In this case the *oldURLParamName* property determines the name of this special request parameter.

In the running example a link wrapper rule is used to realize a security-specific adaptation. Its goal is to deactivate a hyperlink that points from the project homepage to another page presenting information on the project's budget. For users that are logged in as guests, the appropriate rule redirects this link to the login page (see Listing 6.7).

```
<gac:LinkWrapperRule rdf:ID="loginRedirectRule"
                     gac:selector="//a[@href='budget.html']">
    <gac:hasCondition>
      <gac:Condition gac:when="($LogIn=='Guest')"/>
    </gac:hasCondition>
    <gac:toUrl>http://www.gacexample.org/login.html</gac:toUrl>
</gac:LinkWrapperRule>
```

Listing 6.7: Link wrapper rule example

Name	Meaning	Usage	Values
`toUrl`	Name of the new URL	optional	String
`param`	Additional request parameter	optional	name/value pair
`keepOldURL`	Specifies whether the old URL should be retained as a specific request parameter	optional	true|false
`oldURLParamName`	Name of the request parameter containing the old URL	optional	String

Table 6.6: Properties of a link wrapper rule

Note that a link wrapper rule can not only be applied to hyperlink targets but to arbitrary XML attributes describing URLs, e.g. also to the *action* attributes of Web forms. The name link wrapper rule is used because the adjustment of hyperlinks is the rule's most often used application scenario.

6.4.4.6 Sorting Rule

Whereas the rules mentioned above address single content units (XML nodes), there are also rules adapting sets of content units, such as all child elements or all variants of a specific content unit. One of them is the *sorting rule* (Class *SortingRule*) aimed at ordering sets of XML elements according to one of their attributes. The elements to be sorted are addressed by the XPath expression specified by the rule's *selector* property. They are ordered based on the value of the attribute defined by the *sorting rule*'s additional *by* property. This can be either an XML attribute of the selected nodes, or a value of a context data parameter that is parameterized by such an attribute. The *order* attribute dictates whether the ordering is ascending or descending.

Name	Meaning	Usage	Values
`by`	Specifies the attribute according to which the sorting is performed	required	String
`order`	Specifies if the ordering is ascending or descending	required	asc|desc

Table 6.7: Properties of a sorting rule

In our running example the list of project members shown on the project homepage is sorted according to whether (and how often) the user already saw their homepages (see Listing 6.8). The attribute according to which the sorting has to be performed is defined by the adaptation context data parameter called *Visited*. It is an array that is parameterized by the unique identifiers (*id* attributes) of content elements. Members the user was already interested in are shown in the beginning of the list. Therefore, the rules order property has the value *desc*. As no condition is defined, this rule is always executed. Note that this is also an example of adaptivity.

```
<gac:SortingRule rdf:ID="sortMemberRule"
                 gac:selector="//div[@class='member']">
  <gac:by>$Visited[@id]</gac:by>
  <gac:order>desc</gac:order>
</gac:SortingRule>
```

Listing 6.8: Sorting rule example

6.4.4.7 Paginator Rule

A *paginator rule* (Class *Paginator Rule*) aims at dividing sets of XML elements into a number of smaller subsets, each containing only a predefined number of elements. It is typically applied to efficiently place content elements into a subsequent series of grouping elements (e.g. XML elements denoting container-like "grouping" structures such as pages, sections, slices or components), so that only a limited number of content elements is shown in a single group.

The rule's *selector* property addresses the XML elements, the subelements of which should be paginated. As shown in Table 6.8, these subelements are then grouped by grouping elements, the name of which is specified by the additional *group* property. The number of content units in each resulting group is determined by the *number* property. As a matter of course, the size of the last group to be created can be less than *number*, because it contains only the remaining elements.

Name	**Meaning**	**Usage**	**Values**
`group`	Name of the grouping element	required	String
`number`	Number of subelements in each resulting group	required	Integer

Table 6.8: Properties of a paginator rule

6.4.5 Update Rules

Unlike *adaptation rules*, *update rules* (Class *UpdateRule*) aim at updating the adaptation context data. They facilitate to change existing context parameters or to create new ones. Optionally, they can also have a *selector property*. In this case they are triggered for each selected XML node. Otherwise, they are activated only once (i.e. once each time the GAC processes an input document).

The action performed by an Update Rule is specified in its *do* property, a string describing a value assignment to an adaptation context data (ACD) parameter. The *phase* property determines whether the rule is executed before or after the transcoding process. While the *update rules* with the *phase* value *pre* are executed before adaptation rules, those with the *phase* value *post* are processed after them. That is to say, while the effects of an update rule with the *phase* value *pre* can already influence the actual adaptation transformations, the effects of an update rule with the *phase* value *post* can be perceived only at the next page request. Since the usage of the *phase* property is optional, its default value is *pre*[6].

Name	Meaning	Usage	Values
`selector`	selects parts of the input XML content	required	String
`do`	Specifies the action to be performed	required	String
`phase`	Specifies if the update rules is performed before or after the transcoding process	optional	pre/post

Table 6.9: Properties of an update rule

Among the Adaptation Rules of our running example two rules supporting adaptivity were mentioned (see Listing 6.3 and Listing 6.8). Both require to keep track of the content elements already been visited by the user. This update mechanism can be easily supported by the following very simple Update Rule (Listing 6.9).

```
<gac:UpdateRule rdf:ID="trackVisitsRule"
                gac:selector="//div[@id]">
  <gac:do>$Visited[@id]=true</gac:do>
  <gac:phase>post</gac:phase>
</gac:UpdateRule>
```

Listing 6.9: Update rule example

The XPath expression in the rule's selector attribute identifies all content elements by addressing XML elements containing an `id` attribute. The value assignment described in the rule's *do* property tracks the fact that a content element was displayed by appropriately setting the *$Visited* variable. Note that this variable was already referred to in the adaptation rules shown in Listing 6.3 and Listing 6.8. As this rule is not associated with a condition it is always triggered. However, since its *phase* attribute has the value *post*, it is activated only after all other adaptation rules were performed. That is to say, its effect can be perceived only at the user's next page request when the appropriate adaptation rules are performed again.

To demonstrate the "interplay" of update rules and adaptation rules, Listing 6.10 illustrates a more complex adaptation strategy consisting of three rules. The first rule aims at counting a user's page visits in the example application by incrementing the NumberOfClicks variable for each requested page. Again, it has no condition associated, i.e. it is always triggered. If the value stored in the *$NumberOfClicks* variable exceeds the average number of page visits (counted for all user sessions) by a given percentage, the second rule classified the user as "interested in details". In this case the third rule (InclusionRule) includes additional information about the project at the project homepage.

6.5 Implementation Issues

In order to efficiently "reuse" the functionality of the pipeline-based document generation architecture introduced in Section 4.5, the GAC was implemented as one of its transformer

[6]Introduced by the AHAM reference model for adaptive hypermedia applications [De Bra et al. 1999], the phase attribute is widely used in systems supporting adaptivity (see Section 2.2.5).

```
<gac:UpdateRule rdf:ID="clickCounterRule">
  <gac:do>$NumberOfClicks=$NumberOfClicks+1</gac:do>
  <gac:phase>post</gac:phase>
</gac:UpdateRule>

<gac:UpdateRule rdf:ID="trackInterestRule">
   <gac:hasCondition>
     <gac:Condition gac:when="$NumberOfClicks>$AverageClicks*1.5"/>
   </gac:hasCondition>
   <gac:do>$InterestedInDetails='yes'</gac:do>
   <gac:phase>post</gac:phase>
</gac:UpdateRule>

<gac:InclusionRule rdf:ID="includeDetailsRule"
                      gac:selector="//div[@id='project']">
   <gac:hasCondition>
     <gac:Condition gac:when="$InterestedInDetails=='yes'"/>
   </gac:hasCondition>
   <gac:what>http://www.gac.org/AdditionalInformation/</gac:what>
</gac:InclusionRule>
```

Listing 6.10: Interplay of update rules and adaptation rules

modules. As illustrated in Figure 6.12, it can be used at arbitrary stages of the document generation pipeline to perform adaptation transformations on its incoming XML-based input. Each GAC transformer is configured by its RDF-based configuration document. Furthermore, each of them has read and write access to its adaptation context data (ACD), which is a part of the architecture's context model. As shown in Figure 6.12, these parts are typically disjoint. However, the current implementation allows the GACs to access arbitrary parameters from the overall context model.

Since the document generation architecture is based on the publication framework Cocoon, the GAC was developed as a custom Cocoon transformer [Ziegeler and Langham 2002] written in Java. Inheriting from Cocoon's AbstractDOMTransformer class, it performs the corresponding data transformations on the JDOM [@jdom] view of its input XML documents. In order to effectively realize adaptation and context data update rules, a Java class was implemented for each rule type introduced in Section 6.4. The corresponding implementations are optimized for performing the appropriate adaptation operations (elision, separation, inclusion, replacement, sorting, etc.) on the processed XML content.

At configuration time, the GAC processes its RDF-based configuration file and retrieves all rule definitions contained in it. For each retrieved rule it instantiates the corresponding Java rule class and sets its parameters, respectively. Sorted by their priority, the instantiated rules are registered by the *RuleManager*, a Java object maintaining a dynamic array of rule objects. This rule instantiation process is performed only once, i.e. at the time when the Web application is initialized.

At run-time the GAC is triggered by receiving XML content from its preceding data transformation components in the Cocoon pipeline. In this case the *RuleManager* is activated that triggers its registered rules one after another. Utilizing the XPath API, each rule determines the set of XML elements it is assigned to and evaluates whether the corresponding conditions hold. If they hold, the corresponding data transformations or context data updates are performed and the next rule object is invoked. After the last rule is triggered, the resulting XML document is sent to the next processing step in the Cocoon pipeline.

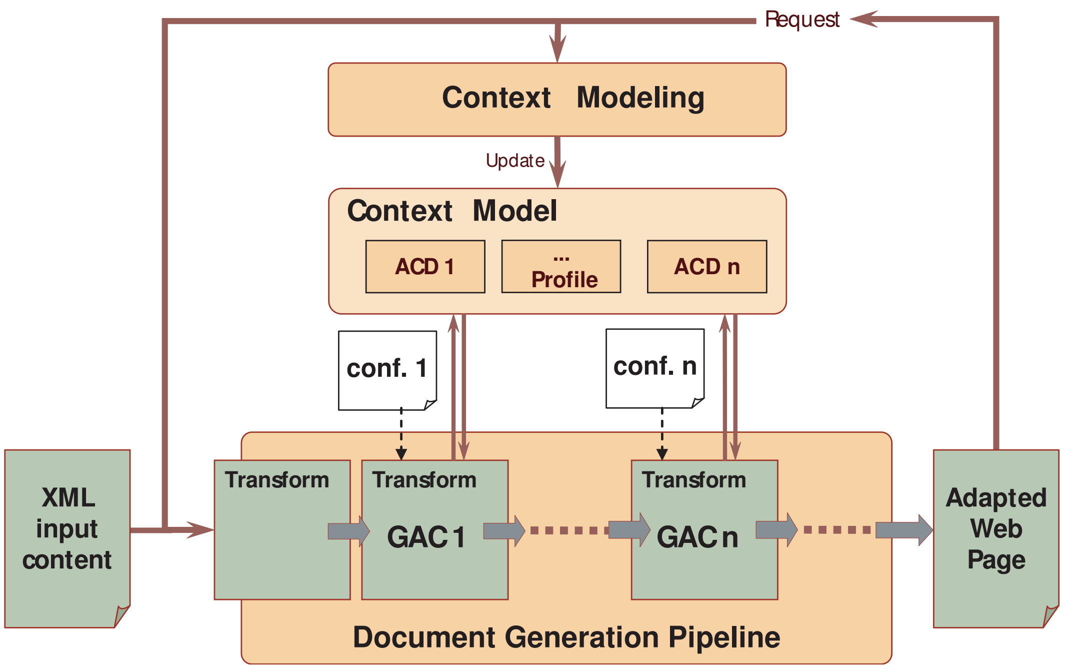


Figure 6.12: GAC implementation overview.

The context model (containing the GAC's adaptation context data) was realized based on Cocoon's so-called authentication context. It allows to store session and context information in form of arbitrary XML data structures that can be manipulated by DOM operations. Furthermore, in a later version, another implementation of the ACD repository based on the open source RDF database Sesame [Broekstra et al. 2002] was also realized . In this version the GAC implementation utilizes SeRQL (Sesame RDF Query language) for retrieving or updating this data. The usage of SeRQL allows for expressing more powerful queries (both select and update operations) on the ACD, as well as the integration of heterogeneous context data sources.

6.5.1 Running Example Implementation Configuration

While Figure 6.12 illustrated the general architecture of a GAC-based implementation, Figure 6.13 depicts the concrete GAC configuration of the running example used throughout this chapter. The input documents of the document generation pipeline are Web pages delivered in form of XHTML documents. They are subdued to two GAC transformers, each of which realizes a certain adaptation concern. For the sake of simplicity, in this scenario both GACs are configured to utilize the same adaptation context data repository.

Since the two GACs are switched immediately behind each other, note that it would be also possible to use only one GAC that is configured by all adaptation and update rules. Still, in order to support a better separation of concerns, the rules executed by each GAC are grouped according to a certain adaptation aspect. The first GAC transformer performs adaptation operations concerning device dependency. The second one performs all other adaptations that deal with user-specific personalization issues. Note that a main advantage of this separation of concerns with different GACs is the possibility to easily "plug-in" or remove a certain adaptation aspect from the entire application. Furthermore, it also nicely

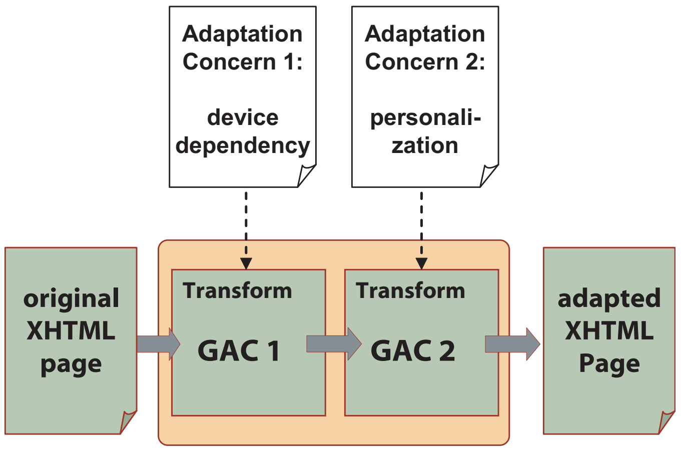

Adaptation Concern 1: device dependency
Adaptation Concern 2: personali-zation
original XHTML page
Transform GAC 1
Transform GAC 2
adapted XHTML Page

Figure 6.13: Running example implementation configuration.

corresponds to the preferred strategy of a Web designer who tries to specify independent application concerns (in this case adaptation issues) separately from each other. The usage of an implementation based on a series of GACs enables to step-by-step incorporate all these additional concerns into the original Web application at run-time.

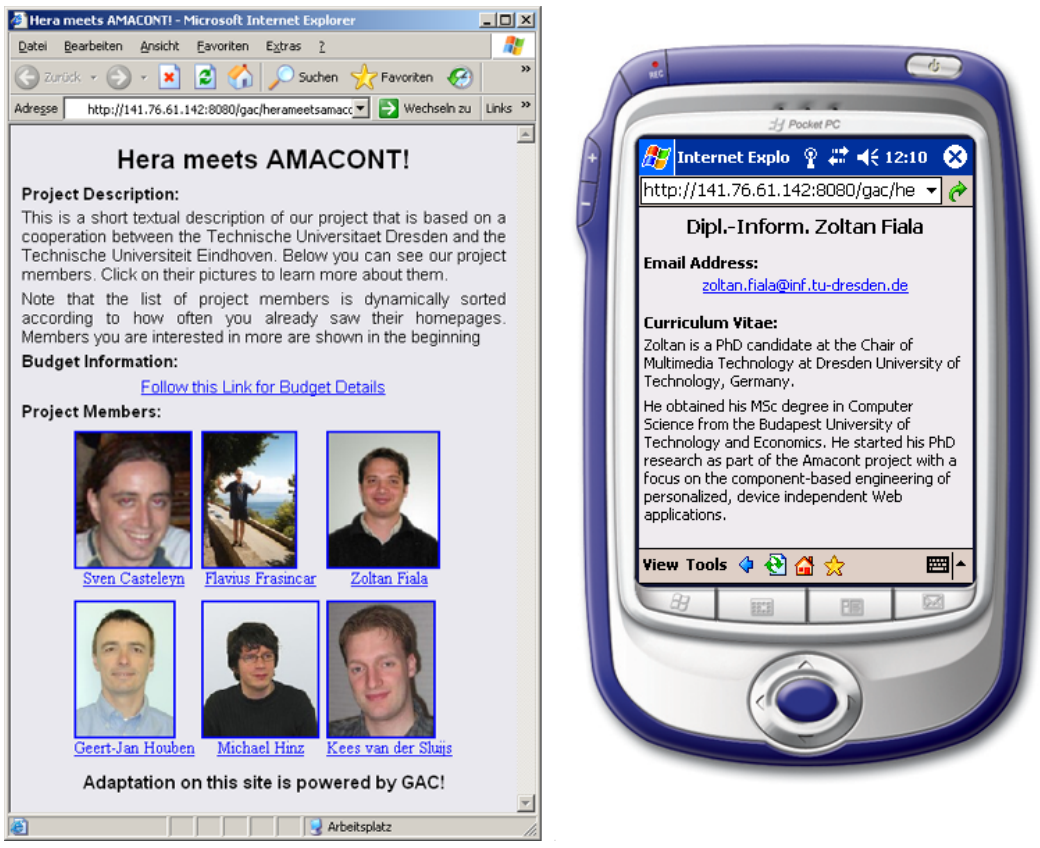
Figure 6.14 depicts two screenshots from the "adapted versions" of the running example. The first one (on the left) shows the project overview page as it is presented on a desktop browser. Note that, according to the inclusion rule described in Listing 6.5, an "advertisement" of the GAC was inserted on its bottom. It indicates that "Adaptation on this site is powered by the GAC". Furthermore, the list of project members was also dynamically reorganized based on the user's visits to their personal homepages. The second screenshot (on the right) shows the Web page of a particular project member on a handheld device. Based on the adaptation rules addressing device dependency, both the image of the project member and the list of his publications was omitted.

6.5.2 Extensibility Issues

The GAC provides a repertoire of generic adaptation operations (rules) that are applicable on arbitrary XML input. Nevertheless, in some cases a designer might require further (e.g. more specific) adaptation rules in order to cope with a given transcoding scenario. The reason for this could be the requirement to specifically target the characteristics of a given XML format (or Web application) by the provision of a set of designated adaptation rules.

As a typical example we mention the well-known "table transform" [Hwang et al. 2003] transcoding operation. It is used for displaying large XHTML tables on handhelds either by unrolling them to a list, or by splitting them to a number of smaller tables (or subtables) with a configurable number of columns and/or rows. These operations can be reduced to a series of basic (generic) transformation operations (inclusion, omission, replacement, etc.). Yet, a provider might need a designated rule for them in order to 1) have a more "high-level" view on these operations or 2) to achieve a better performance by taking into account the specific characteristics of the given XML format (in this case XHTML) and putting all required functionality in one rule.

To realize such extensions two steps have to be performed. First, a new GAC adaptation rule (and its possibly parameters) have to be specified in the GAC rule schema. Second,


Hera meets AMACONT! - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten
Adresse http://141.76.61.142:8080/gac/herameetsamacc
Wechseln zu Links
Arbeitsplatz
Pocket PC
Internet Explo 12:10
http://141.76.61.142:8080/gac/he
View Tools

Hera meets AMACONT!

Project Description:

This is a short textual description of our project that is based on a cooperation between the Technische Universitaet Dresden and the Technische Universiteit Eindhoven. Below you can see our project members. Click on their pictures to learn more about them.

Note that the list of project members is dynamically sorted according to how often you already saw their homepages. Members you are interested in more are shown in the beginning

Budget Information:

Follow this Link for Budget Details

Project Members:

Sven Casteleyn, Flavius Frasincar, Zoltan Fiala, Geert-Jan Houben, Michael Hinz, Kees van der Sluijs

Adaptation on this site is powered by GAC!

Dipl.-Inform. Zoltan Fiala

Email Address:

zoltan.fiala@inf.tu-dresden.de

Curriculum Vitae:

Zoltan is a PhD candidate at the Chair of Multimedia Technology at Dresden University of Technology, Germany.

He obtained his MSc degree in Computer Science from the Budapest University of Technology and Economics. He started his PhD research as part of the Amacont project with a focus on the component-based engineering of personalized, device independent Web applications.

Figure 6.14: Running example screenshots

the appropriate adaptation operations to be performed on the input XML content have to be implemented. Since the extensible rule classes of the current GAC implementation allow to utilize DOM operations on the input content, programmers can easily add arbitrary functionality, either by programming it themselves or by importing existing DOM-based data transformation implementations. Furthermore, we note that the DOM (JDOM) libraries used for the GAC's implementation also allow programmers to apply any arbitrary existing XSLT stylesheet on the DOM tree of the actual document.

Besides adaptation rules, a Web developer using the GAC might also need further functionality for updating the adaptation context data. While the current GAC update rules are quite generic, we note that they are rather low-level and do not support for the expression of more specific context modeling (e.g. self-learning) algorithms. Again, the use of a Java-based implementation and the standardized CC/PP-based interface to the adaptation context data allow to easily incorporate existing context modeling components into the GAC's rule repertory.

6.6 Conclusion and Discussion

This chapter introduced the GAC, a generic transcoding tool for making XML-based Web applications adaptive. Based on the key observation that an Adaptive Web Information System is typically realized as a series of transformations, it was shown how it can *add* additional adaptation concerns to an existing Web application without the need to completely redesign it. First, an overview of the GAC's main functionality was given, and a number of its possible application scenarios were demonstrated. Then, an RDF-based rule language for the specification of the GAC's exact functionality was presented. The different kinds of content adaptation and context data update rules were illustrated based on a running example. Finally, a prototypical implementation (developed by the author of this thesis) for realizing the GAC's main functionality was presented.

As could be demonstrated, the current GAC architecture provides the necessary functionality to incorporate different kinds of adaptation into XML-based Web applications. Still, the author is aware of the fact that, on top of this foundation, there is a need for concepts and visual GAC configuration tools that would allow Web developers to specify such additional adaptation concerns in a high-level systematic way. This issue of "GAC-based authoring" was not a focus of this work and has thus not been sufficiently addressed, yet. We note, however, that initial ideas in this direction will be discussed within the scope of future work ideas presented in Section 7.3.

Nevertheless, an interesting aspect to be discussed here is the relation of the GAC to the concern-oriented component model presented in Chapter 4. Even though the GAC was inspired by the adaptation functionality of that model and its pipeline-based document generation architecture, note that it pursues a complementary approach. Whereas a Web presentation built of document components contains adaptation definitions from the beginning in a component-based inherent way, in a Web transcoding scenario this adaptation is added to the underlying application afterwards in a rule-based manner. As a matter of course, both solutions have a number of advantages which mainly depend on the respective application scenario. The rest of this section is dedicated to the discussion of these differences in more detail.

Adaptation by Transcoding: Advantages

In general, the separate specification, storage, and implementation of "external" adaptation rules has the following advantages.

Adaptation support without content reauthoring: The providers responsible for adaptation do not need to manipulate or reauthor the input content in order to prepare it for adaptation. That is to say, XML documents from arbitrary content authors can be taken as input. Furthermore, the provider of the transcoding solution does not need to be granted write access to this original content.

Adaptation support for future Web applications: Adaptation rules may be specified also for Web pages (or in more general for Web content) that have not been even created, yet. As an example, a GAC configurator might create a rule dictating that all images in any incoming XHTML document have to be elided for handheld devices. As a matter of course, this rule is applicable for all kinds of dynamically created volatile XHTML input, too.

Flexible adaptation reconfiguration: As a consequence of the former advantage, the same input document can be used in different adaptation scenarios. It is merely the configuration of the transcoder that has to be altered in order to change the current adaptation policy. For instance, while a configuration $C1$ might be used to adjust XHTML documents to mobile devices with small displays, another configuration $C2$ could be used to transcode them for users with visual impairments. Since the corresponding adaptation specification are not intertwined with the content to be adapted, it is easily possible to add new ones or remove existing ones without having to redesign the original application. Furthermore, the possibility to switch several transcoders in line also allows to easily change the order or priority of the applied adaptation concerns. Thus, instead of using "fixed adaptations" that are "hard-wired" to the original content, the configurator of the GAC can "adapt the adaptation" to the particularities of the given transcoding scenario.

Independent application and adaptation evolution: Due to the separate storage of transcoding rules from the original content, the introduction of a new GAC rule type or the modification (or extension) of an existing one does not require to change all corresponding input documents. Furthermore, a change to a given rule's implementation (e.g. for the sake of performance enhancements) affects only the "inner life" of the GAC, not necessitating to change the entire Web application to be adapted. That is to say, the specification and implementation of adaptation rules may evolve independently from the input documents.

Support for distributed adaptation operations: The flexible assignment of adaptation rules to documents' parts by XPath expressions allows to attach rules to multiple fragments of a document. Thus, a single adaptation rule be used to adjust different parts (components) of a Web page. This advantage is especially important because adaptation concerns (e.g. the omission of high quality pictures for devices with low presentation capabilities) are typically not pinpointed to a specific element of the input content, rather spread over several similar content elements (e.g. in this case all appropriate images) in a Web application. Consequently, such an adaptation rule addressing a number of content elements can be easily added, removed, or altered by changing only a small part of the separately stored and managed adaptation configuration.

Adaptation by Transcoding: Limitations

On the other hand, an adaptation scenario based on a transcoding solution utilizing external adaptation rules has also some limitations.

Adaptation by content filtering: The basic principle of the GAC is to perform context-dependent transformations on an already existing content stream. Thus, the adaptation operations supported by it are "restricted" to filtering and/or reorganizing this input content, not supporting to easily add new content alternatives. Even though *inclusion rules* allow to insert XML-fragments into the processed Web documents, the introduction of a new adaptation variant (e.g. a video representation of all products of an online shop for users with high bandwidth connections) typically requires a more thorough reengineering of an existing Web application.

Need for detailed knowledge of the input XML content: The efficiency of a transcoding solution significantly depends on how much the "configurator" of transcoding rules knows about the input content. Generally, the more he is familiar with the structure (e.g. the underlying data model or schema) of the input documents, the more powerful adaptations he can to express. Still, this configurator is often independent from the author(s) of the original application. Furthermore, modern Web applications are increasingly developed with high-level visual authoring tools that aim at hiding the rather low-level XML notation of the the underlying engineering approach. On the contrary, if a Web author (or designer) considers adaptation as an inherent issue of the

Web application to be created from the beginning, an appropriate process model can optimally support him.

Dependence on the semantic richness of the input content: Another important question is at which stage of the overall data transformation process a transcoding tool can be utilized. As stated above, its efficiency mainly depends on how well structured the input data is, especially how much metadata it contains. Increasingly, modern Web Information Systems utilize XML-based representations of the data they process. Starting from a well annotated data structure, they perform a number of transformations leading to a hypermedia presentation. Of course, if the data transformation pipeline cannot be "cut up" and the transcoder can only operate on top of the final presentation, then its adaptation capabilities are mostly restricted to presentation adaptation.

Dependence on the underlying architecture: As a consequence of the previous issue, the applicability of the GAC also depends on the overall architecture of the underlying Web application. That is to say, the question of where and how efficiently the GAC can be used is significantly influenced by the modularity and extensibility of that architecture. Furthermore, we mention that modern Web applications are typically developed and maintained by complex engineering frameworks, application servers, content management systems, etc. As a matter of course, a possible GAC-based extension has to be in "perfect harmony" with all these architecture components.

Low-level adaptation specification: The usage of a transcoding tool (e.g. the GAC) requires a rather low-level specification of adaptation transformations in terms of adaptation rules that operate on XML elements. Furthermore, since the GAC and its configuration language are by definition independent of a given XML grammar or methodology, it is also difficult to provide a generic graphical authoring tool for intuitively adding adaptation to any existing Web application in a high-level manner. Consequently, such a graphical tool has to be created separately (e.g. as an "add-on" or a "plug-in") for each specific authoring tool.

No inherent support for type safety: In principle, external transcoding rules can perform arbitrary transformations of the input content. Nevertheless, it might be the case that different adaptations (such as the omitting of a content element or XML tag) lead to invalid documents, i.e. to documents that already do not correspond to their original data schema. This might lead to problems when the affected documents are processed by further transformation steps, e.g. in order to be presented in a given output format. On the other hand, a component model with built-in support for adaptation permits only document transformations that assure type safety and validity.

Lacking support for component-based reuse: Separating adaptation rules from the underlying application prevents the efficient reuse of adaptable implementation artefacts in a component-wise, black-box-like manner. Reusing a part of the base application in another composition scenario also implies to extract its corresponding adaptation recipe from the original application's adaptation configuration and to "transfer" it to the new application's adaptation configuration in a possibly modified way. Thus, the detached management of application and adaptation code can lead to higher maintenance efforts.

Orphan adaptation rules: Since in a transcoding scenario the adaptation rules are fully detached from the input content, they also have to be maintained separately. However, this separate storage of external transcoding rules (e.g. GAC rules) might lead to inconsistency problems if changes to the structure of the input documents (i.e. the documents to be adapted) are made. For instance, if the XPath expression used as the selector of a given GAC rule addresses e.g. the second subelement of a given XML element, then the inclusion of a new preceding sibling can lead to an unexpected transcoding effect. Especially, such inconsistencies arise if the person modifying a Web application is not familiar with its corresponding GAC-based adaptation configuration.

However, this problem can be reduced to the problem of so-called *orphan annotations* [Brush et al. 2001] (i.e. external annotations that can no longer be attached to a document because it was modified) known from Web annotation systems. While not being a central issue of this work, we note that there have been several approaches proposed that address this problem by utilizing so-called *robust annotation positioning* techniques. For more information on this topic the reader is referred to [Brush et al. 2001, Abe and Hori 2003].

To sum up, the separation of adaptation from the original application provides for more flexibility in terms of reconfiguration and evolution (adding, removing, or reordering adaptation aspects, inventing new adaptation rules, etc.), and the possibility to define adaptation conditions (operations) that are distributed over multiple content elements while using only one adaptation rule. On the other hand, a solution based on adaptive document components provides better reusability (in terms of adaptable content artefacts), consistency, validation support, as well as the possibility to use dedicated high-level design and authoring tools.

We remark furthermore, that a combined usage of both approaches is also conceivable. It is possible to design and implement an adaptive Web application based on reusable document components and, if demanded, flexibly add additional and rather "volatile" adaptation concerns (e.g. adaptations that are only needed in a specific deployment of the application) to it based on one or more GACs. In this case the GACs mainly serve as "customizers" that make the application runnable in a given scenario that was not foreseen at the time of its original design. Thus, while a component-based approach provides for "fixed" adaptations, GACs can be utilized for further "adapting these adaptations". Note that the modularity of the component-based document format's (staged) document generation architecture allows to plug-in GAC components at any arbitrary stage of the data transformation pipeline.