Developing Component-based Adaptive Web Applications with the AMACONTBuilder

Zoltán Fiala, Michael Hinz, Klaus Meissner
Dresden University of Technology
Heinz-Nixdorf Endowed Chair for Multimedia Technology
D-01062 Dresden, Germany
{zoltan.fiala, michael.hinz, kmeiss}@inf.tu-dresden.de

Abstract

The growing need for personalization and device independence calls for effective ways of engineering adaptive Web applications. This requires formats, languages and structured process models that allow developers to design and express adaptation, but also appropriate authoring tools. Still, current Web authoring tools do not provide sufficient adaptation support. Recently, the AMACONT project introduced a component-based document format aiming at efficiently composing adaptive Web applications from reusable Web components. This paper focuses on the development of such applications and presents a visual authoring tool called AMACONTBuilder. Based on a structured authoring process it is shown how it can be utilized for effectively creating AMACONT applications.

1. Introduction

The WWW's evolution to a ubiquitous medium offering personalized dynamic information calls for new ways of effectively engineering adaptive Web applications. Still, current document formats (XHTML, cHTML, WML etc.) are hardly suitable for this purpose. The main reason for this is the missing separation of concerns (such as content, layout, navigation) as well as the lack of mechanisms for expressing adaptation in a generic way.

To address this problem, the AMACONT project introduced a component-based XML document format [5] that enables to compose adaptive Web applications by the aggregation and linkage of reusable document components. Furthermore, a modular document generation architecture for dynamically adjusting adaptable components to different user preferences, client devices and Web formats was developed.

Since the development of component-based adaptive Web applications is a complex challenge, it has to be based on a structured authoring process considering different aspects of adaptation. In recent work we adopted the model-driven Hera design methodology for this purpose [6, 4]. Still, the utilization of intuitive graphical development tools is crucial for the success of the overall authoring process. Therefore, this paper introduces the AMACONTBuilder, a visual authoring framework for the efficient creation of component-based adaptive Web applications.

The paper is structured as follows. After addressing related work in Section 2, the document model of AMACONT (Section 3) as well as the main concepts and the overall architecture of the AMACONTBuilder are described (Section 4). Then, Section 5 describes the structured development process of component-based adaptive Web applications with the aid of the AMACONTBuilder in detail. Finally, Section 6 illustrates the stepwise publishing process of the resulting Web presentations.

2. Related Work

Recently, different tools for authoring adaptive hypermedia and Web applications have been developed. As a significant approach we mention the AHA! [1] software platform that provides a reference implementation of the AHAM reference model [2]. Based on Java applets, it provides graphical tools for defining high-level concepts, concept relationships and adaptation rules. However, as AHA! allows only for authoring on the conceptual level, there is no support for creating Web resources (e.g. media elements or page fragments) assigned to concepts. Furthermore, AHA! does not allow for developing data-driven adaptive Web applications.

Another important group of visual Web engineering software solutions are CASE tools aiming at modeling data-driven Web Information Systems (WISs [11]). They are based on structured design methodologies that typically distinguish between the *conceptual*, *navigational* and *presentation model* of a Web application. As prominent examples

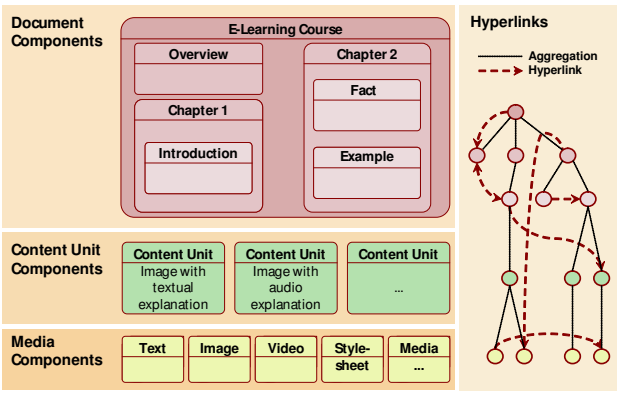

Figure 1. The Document Model.

we mention WebRatio (based on the modeling language WebML [3]), VisualWade (based on the OO-H method [7]), ArgoUWE (based on the UWE methodology [13]) and the HPG tool [14] of the Hera specification framework [15]. Since Web Information Systems provide a hypermedia view on highly-structured dynamic data sources, these modeling tools operate on the schema view of the input data and specify WISs in terms of *abstract* concept, navigation and user interface elements. The creation and configuration of concrete content assets is out of their scope. Furthermore, there is insufficient support for the visual configuration of the adaptive behavior of Web site components, yet.

The AMACONTBuilder presented in this paper follows a different approach. Firstly, it supports the visual development of adaptive Web presentations on the instance level. Furthermore, by utilizing the "programming by example (PBE)" paradigm it also allows for generalizing authoring operations performed on instances for data-driven AMACONT component templates, thus allowing for authoring data-driven adaptive Web presentations. Moreover, a main focus is on the visual configuration of adaptation in different stages of the authoring process.

3. AMACONT Component Architecture

The component-based document format of AMACONT [5] allows to build device-independent Web applications by aggregating and linking configurable document components. These are documents or document fragments, instances of an XML-grammar representing adaptable content on different abstraction levels (see Figure 1).

On the lowest level there are *media components* encapsulating concrete media assets. These comprise text, structured text (e.g. HTML), images, sound, video, Java applets etc. The second level combines media components belonging together semantically - e.g. an image with a textual description - into so called *content unit components*. Defining such collections is a key factor of reuse. The spatial adjustment of contained media components is described by client-independent layout properties (see Section 3.2). Thirdly, *document components* are specified as parts of Web presentations playing a well-defined semantic role (e.g. a news column, a product presentation or even a Web site). They can either reference content units, or aggregate other document components. The resulting hierarchy describing the logical structure of a Web site is strongly dependent on the application context. Finally, the orthogonal hyperlink view defines links spanned over all component levels. Uni- and bidirectional typed hyperlinks based on the standards XPath and XPointer are supported.

3.1. Describing Adaptive Behavior

To define adaptive behavior in a generic way, each component may include a number of variations. As an example, the definition of an image component might include (in its body) two variants for color and monochrome displays. The decision, which alternative is selected, is made during document generation according to a selection method contained in the component's header. Such selection methods are chosen by component authors and can represent arbitrarily complex conditional expressions parameterized by user model parameters. The XML-grammar for selection methods allows the declaration of user model parameters, constants, variables, and operators, as well as complex conditional expressions of arbitrary complexity [5]. A concrete example for defining such selection methods will be shown in Section 5.1.

3.2. Layout Adaptation

As already mentioned, the document format allows to describe the spatial adjustment of components by client-independent layout properties. Inspired by the layout manager mechanism of the Java language, they describe a size and client-independent layout allowing to abstract from the exact resolution of the display or the browser's window. The exact rendering of media objects is done by XSLT stylesheets transforming these abstract layout descriptions into concrete output formats. At current time four layout managers are defined: $BoxLayout$, $BorderLayout$, $OverlayLayout$ and $GridLayout$. A number of stylesheets for automatically converting such descriptions to different Web formats were developed. For more details on AMACONT's layout managers the reader is referred to [4].

3.3. Document Component Templates

The document components introduced above are static, i.e. they represent a concrete piece of Web content, such as a specific instance of an image (media component) or a chapter in an eLearning course (document component). Still, in order to provide support for data-driven Web applications, like online-shops, e-galleries etc., there is a need for creating components on the fly from dynamic data sources. For this reason so called *document component templates* have been developed [6]. These are component skeletons declaring the structural, behavioral and layout aspects of components independent of their concrete content. They are instantiated by being filled with media instances dynamically queried (retrieved) from a data source.

4. AMACONTBuilder: Main architecture

The AMACONTBuilder was implemented in Java and is a modular authoring tool allowing for creating and editing arbitrary XML documents. It is based on a generic framework that can be extended by graphical editor plug-ins for visually authoring specific types of XML content.

In order to provide programmatic access to all kinds of XML data, the AMACONTBuilder utilizes a flexible internal object model which is based on JDOM. This generic object model was extended by AMACONT specific classes providing an API for efficiently manipulating adaptive Web components. That is to say, AMACONT documents are parsed into a hierarchy of component specific objects when they are opened by the AMACONTBuilder. As an example, the developer of an editor plug-in for image components can utilize predefined Java methods for getting and setting image metadata, for creating image component variants and selection methods etc.

Note that the utilization of such an object model has different advantages. Firstly, plug-in programmers can use a high-level API for manipulating AMACONT components and do not have to bother about their concrete underlying XML-based format. Secondly, this solution provides also more robustness regarding to modifications of the underlying XML languages. During the evolution of the AMACONT project different changes to the component model's XML-based description language have been made, especially in order to provide less redundant descriptions and better performance in the document generation process. Still, as the plug-ins of the AMACONTBuilder work on an internal object model, it was enough to adjust the mappings between that model and the XML-based formats, not needing to modify the application logic of specific plug-ins.

As shown in Figure 2, the user interface of the AMACONTBuilder consists of different parts. The *application frame* provides generic functionality for configuration options as well as for file and object model management. It is responsible for parsing XML files into the internal object model, for assigning editor plug-ins to parts of it, and for serializing the modifed object model to XML, respectively.

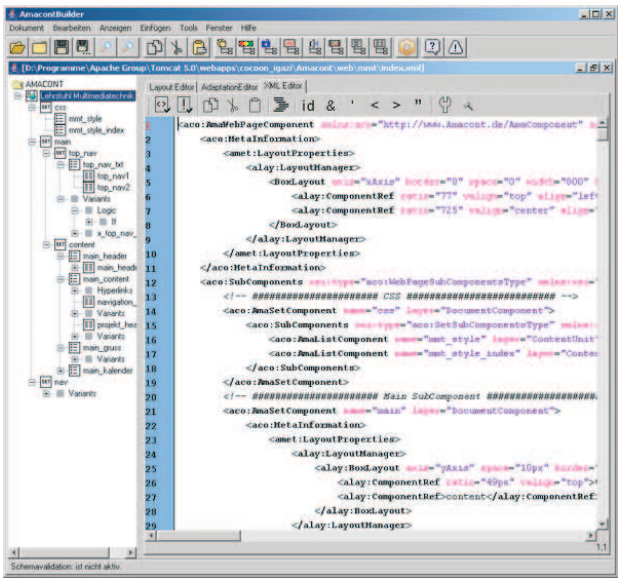

Figure 2. AMACONTBuilder.

The *navigation frame* on the left provides a tree-based view on the node (component) structure of the currently edited XML document. When navigating through these components, the editors assigned to the appropriate component types are activated in the editor frame. Note that different filters have been implemented for the navigation tree. For instance, authors have the possibility to see only image components.

Finally, the *editor frame* (on the right) provides space for the actual editor plug-ins associated to different node (or component) types[1]. The assignment of editors to node (component) types is determined by an XML-based configuration file. Whenever there are several editors assigned to a given component type, they are arranged by different tabbed panes in the editor frame.

Whereas there are editor modules that are applicable to all kinds of XML content (e.g. the XML code editor shown in Figure 2), most modules are assigned only to specific node (component) types and thus activated at well-defined phases of the overall authoring process. The structured development process of component-based adaptive Web applications aided by the AMACONTBuilder is described in the following section.

1 Such plug-ins can be implemented by inheriting from the plugin class *de.tudresden.inf.amacont.plugins.AbstractEditor*.

5. Development with the AMACONTBuilder

The development of component-based adaptive Web applications is a complex process that has to be based on structured process models. In previous work [6] we adopted the Hera design methodology for this task, furthermore, we also showed how existing high-level Hera design artefacts can be automatically transformed to a component-based AMACONT implementation [4]. In this section we concentrate on the development of component-based adaptive Web applications "from scratch" and illustrate how the AMACONTBuilder can be used to systematically create AMACONT applications based on a structured authoring process.

Our running example is a dynamic Web presentation providing information about our research group's members. It is part of an AMACONT application presenting our faculty's *media informatics* study program. Research members are presented by their attributes, e.g. their names, contacts, CVs, pictures as well as links to separate pages presenting their publications. To support personalization and device-independence different adaptations are provided.

5.1. Content Authoring

The first step of the authoring process focuses on creating content elements to be presented in the resulting Web application [6]. This implies to create concrete content elements (text, image, audio components etc.) or to define and set up a structured data source from which content elements can be retrieved. Since there exist efficient tools for modeling application domains and creating corresponding data schemas, the AMACONTBuilder focuses on the creation of media components or media component templates.

For this purpose different visual media component editors (text editor, image editor, video editor, audio editor and CSS editor) have been created. Instancing our research group's logo, Figure 3 presents the image editor allowing to upload images in different formats, to edit their properties in a visual way and to save them as image components. The size of an image can be altered either by mouse dragging or by setting the explicit pixel values by input fields.

There are different adaptation issues to be considered at content authoring. Firstly, it is necessary to create content alternatives with different media quality [6]. For instance, in a device independent Web presentation it is necessary to provide different instances of a certain picture with variable size, color depth or resolution in order to automatically adapt to various display types. Another important content adaptation aspect is internationalization. Depending on the nationality of the targeted audience the content assets have to be available in different languages.

Therefore, the content editors were extended with generic mechanisms for creating content alternatives. For

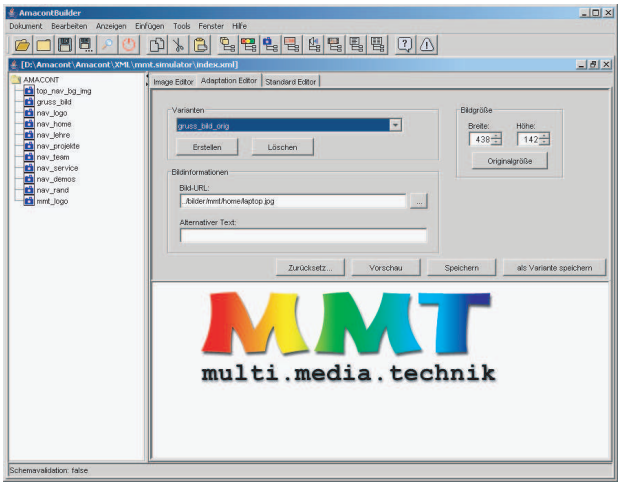

Figure 3. Image Editor.

instance, in the image editor it is possible to provide an alternative text for browsers that are not able to present images. Furthermore, image variants with different quality alternatives can be added, too. Such variants can be created in three ways: by uploading alternative images, by resizing the current image and save it as a new variant or by generating new images automatically. In the latter case the author can predefine the properties (e.g. pixel size, color depth, image format) of an arbitrary number of variants to be created. According to his configuration, the alternative media instances are generated automatically. This feature was implemented by using the Java API of ImageMagick [10].

After creating content alternatives, authors can attach *appearance conditions* to them in order to specify under which circumstances they should be included in the generated presentation. These are Boolean conditions referencing parameters from the context model which is based on CC/PP [12]. As shown in Figure 4, authors can visually navigate through the hierarchy of profiles, choose the appropriate parameters and insert them into appearance rules. The example declares to use the current picture only for browsers with at least 8 bits per pixel color depth and more than 400 px horizontal resolution. Following code snippet illustrates the resulting XML-based selection method (see Section 3.1).

```
<aada:Expr>
  <aada:Term type="and">
    <aada:Term type="lt">
      <aada:UP>ScreenSizeX</aada:UP>
      <aada:Const>400</aada:Const>
    </aada:Term>
    <aada:Term type="lt">
      <aada:UP>BitsperPixel</aada:UP>
      <aada:Const>8</aada:Const>
    </aada:Term>
```

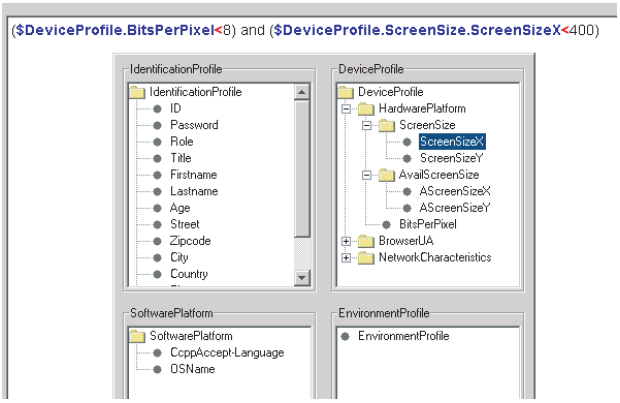

Figure 4. Defining Appearance Conditions.

```
  </aada:Term>
</aada:Expr>
```

The mechanisms described above aim at authoring adaptable content instances. Still, in order to support for data-intensive Web applications, the editor tools can be switched from this "instance mode" to the so called "template mode". As an example, while the logo depicted in Figure 3 is an image component instance being the same on all generated pages, the image presenting the currently visited research member's photo is dynamic and has to be realized as a component template. Therefore, the image editor can be connected to a dynamic data source by a query selecting images and their attributes. Consequently, the resulting code is a component skeleton that can be filled with dynamically retrieved images on-the-fly. The user of the image editor has the possibility to navigate through the result set delivered by the query and to perform modifications both on instance and template level. Firstly, he can edit the currently selected instances so that the altered attributes are written back to the connected database. Secondly, he can also do changes on template level by e.g. assigning an alternative data source containing the PDA variants of our research members' photos and defining a corresponding selection rule.

5.2. Hypertext Authoring

Hypertext authoring (or navigation authoring) aims at describing the logical, structural and navigational aspects of component-based adaptive Web applications [6]. The content assets (or templates) created in the content authoring step are combined to content units or document components (or their corresponding templates). Components can be both recursively aggregated (i.e. include other components) and connected via hyperlinks. Still, note that during hypertext authoring only the nesting hierarchy of components is determined, their visual arrangement is specified in a later step.

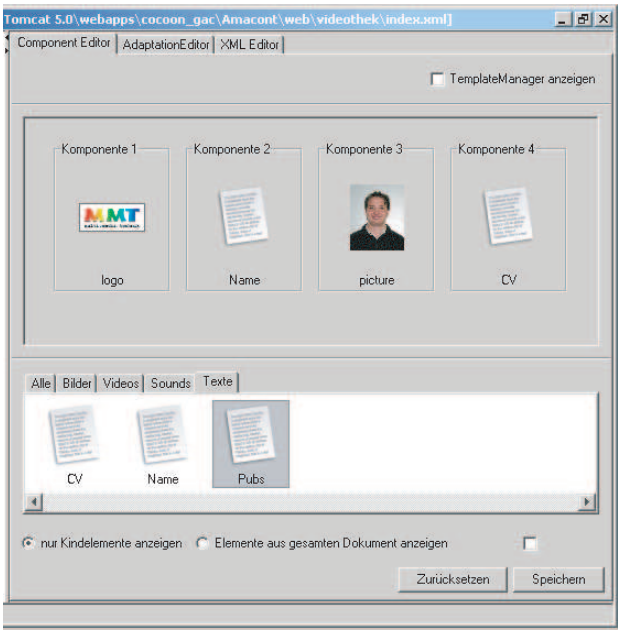

Figure 5. Structure Editor.

For visually creating component hierarchies the *structure editor* was created (see Figure 5). The available content components (or templates) shown on the bottom of the editor tool can be easily aggregated to hierarchies by simple "Drag&Drop" mechanisms. As depicted in Figure 5, a document component describing research members is put together from several subcomponents: the logo of our research group, the group member's name, picture, contacts and CV. Note that hyperlinks can be authored as specific content elements.

By double-clicking on a component in the structure editor the appropriate editors assigned to the chosen subcomponent are automatically activated. For instance, by activating the picture depicted on the left of the component structure edited in Figure 5, the image editor shown in Figure 3 is invoked in a modal editor window.

As a matter of course, the structure editor can be switched to template mode, too. What is more, it can aggregate both static component instances as well as dynamic component templates. In Figure 5 the textual description and the image describing research group members are dynamic templates, i.e. different for each member instance. On the other hand, the logo of our research group is constant for all members and is therefore a component instance.

Adaptation plays a very important role during hypertext authoring, too. According to varying characteristics and (even dynamically changing) preferences of users and their

client devices, different component and hyperlink structures are conceivable. Firstly, it is meaningful to adapt the coarse navigational structure by defining variants of document components providing different views on the underlying content. Secondly, the population of each specified component with content assets can be personalized, too. According to the media preferences and/or devices capabilities of different users, different media types for presenting the same concept can be utilized. As an example, take the case of two users, one of them preferring multimedia content, the other rather textual information. Whereas the first one could be shown a video or audio sequence about a research member's CV, the second one should be provided with a detailed textual description.

Similar to content authoring, adaptation at hypertext authoring can be specified by defining alternative component structures and selection methods. Users can create alternative component structures either from scratch or just do some modifications to an existing structure and save the changed version as a variant. Since the definition of variants and appearance rules was implemented as a generic service provided by all AMACONT editor modules, the same mechanisms as described in Section 5.1 can be utilized.

5.3. Presentation Authoring

Presentation authoring aims at declaring the "look-and-feel" of a Web application independent from its implementation [6]. Complementary to hypertext authoring, where the developer is concerned with organizing the overall presentation structure, presentation authoring specifies how components should be displayed. Moreover, the corporate design of the resulting Web application (determined by layout elements such as font sizes, logos, buttons, background images etc.) is also defined in this step. Therefore, two specific editor modules, the layout editor and the CSS editor, have been developed.

As mentioned in Section 3.2, the component-based document format provides an XML-based layout manager mechanism for specifying the spatial adjustment of subcomponents within their container components in a size and client-independent way. The layout editor shown in Figure 6 allows to attach such layout managers to components and to visually configure their attributes. Layout managers are visualized as a grid that can be filled by icons representing subcomponents. Various "Drag&Drop" techniques have been realized in order to perform most operations graphically, such as resizing the grid, placing subcomponents into grid cells, changing their alignment etc. Besides, various input fields for fine-tuning all layout attributes can be found on the right editor pane. Furthermore, a preview function for XHTML was developed, too.

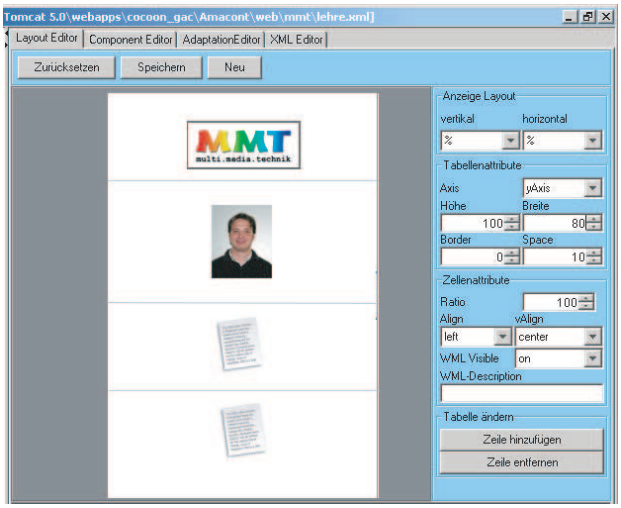

Figure 6. Layout Editor.

Figure 6 depicts the layout of our running example designed for a PDA. The content pieces describing research members are arranged in a vertical *BoxLayout*. Even though the figure depicts a "research member instance", the resulting component is saved as a dynamic template.

On the other hand the CSS editor aims at defining the design of the resulting application. It is a simple authoring module allowing for loading CSS files as well as for maintaining their style definition entries. The resulting definitions are stored as AMACONT CSS components.

Adaptation at presentation authoring focuses on the spatial adjustment of layout elements. Depending on the screen size, the supported document formats and interaction techniques provided by different client devices, these presentation components should be displayed variably. As an example, consider our *BoxLayout* defined for PDAs (see Figure 6). For a desktop PC with a bigger horizontal resolution a "nicer" *BorderLayout* is used. A further adaptation target is the corporate design (the look and feel) of the Web presentation. As an example, in an online shop it is usual to provide varying design variants according to different user properties (age, education, interests, visual impairments etc.) but also according to external parameters (seasons, events, anniversaries etc.). For a more detailed description of possible adaptation issues at presentation authoring the reader is referred to [4].

Note that the specification of adaptation issues is again based on the definition of alternatives and corresponding selection methods.

6. Publishing Process

Document generation is based on a stepwise pipeline concept (see Figure 7). The inputs of the document genera-

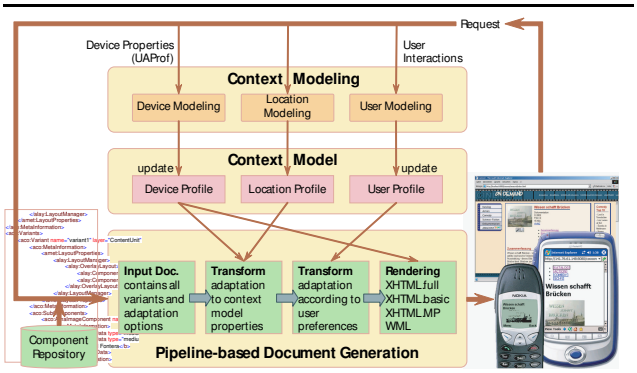

Figure 7. Publishing Process.

tor are complex document components or templates created by the AMACONTBuilder that encapsulate all possibilities concerning their content, layout, and structure. According to the context model (which is stored on the server according to the CC/PP [12] standard), they are subdued to a series of Java and XSLT transformations, each considering a certain adaptation aspect by the configuration and selection of component variants. The document generation architecture of AMACONT is based on the XML publishing framework Cocoon [16]. It provides generic mechanisms for acquiring and processing context information [9], thus allowing to dynamically react on the user's navigation and interaction behavior. For further information on our system architecture the reader is referred to [8].

7. Conclusion and Future Work

The development of component-based adaptive Web applications is a complex challenge that has to be based on a structured authoring process aided by visual authoring tools. Therefore, this paper presented the AMACONTBuilder, a visual authoring tool for creating adaptive Web components. The main architecture of the AMACONTBuilder was presented, and the development process of adaptive Web applications with its aid was explained. Finally, the publication process of the resulting implementation artefacts was illustrated.

Ongoing work on the AMACONTBuilder aims at implementing new editor modules, especially for the visual authoring of adaptable hyperlink structures as well as for the definition of interaction elements on the basis of the W3C XForms standard. Furthermore, as the formats and the generation architecture of AMACONT support generic means for acquiring and processing context information and user interactions [9], new means of visually configuring this dynamic behavior of component-based adaptive Web applications will be designed and implemented.

References

[1] P. D. Bra, A. Aerts, D. Smits, and N. Stash. Aha! version 2.0, more adaptation flexibility for authors. In *AACE ELearn'2002 conference*, pages 240–246, 2002.

[2] P. D. Bra, G. J. Houben, and H. Wu. AHAM: A dexter-based reference model for adaptive hypermedia. In *10th ACM Conference on Hypertext and Hypermedia (HYPERTEXT '99), Darmstadt, Germany*, pages 147–156. ACM, 1999.

[3] S. Ceri, P. Fraternali, and A. Bongio. Web modeling language (WebML): a modeling language for designing web sites. In *9th International Conference on the World Wide Web (WWW9), Amsterdam*, 2000.

[4] Z. Fiala, F. Frasincar, M. Hinz, G.-J. Houben, P. Barna, and K. Meissner. Engineering the presentation layer of adaptable web information systems. In *Fourth International Conference on Web Engineering (ICWE2004), Munich*, 2004.

[5] Z. Fiala, M. Hinz, K. Meiner, and F. Wehner. A component-based approach for adaptive dynamic web documents. *Journal of Web Engineering, Rinton Press*, 2(1&2):058–073, September 2003.

[6] Z. Fiala, G.-J. Houben, M. Hinz, and F. Frasincar. Design and implementation of component-based adaptive web applications. In *19th Symposium on Applied Computing (SAC2004), Nicosia, Cyprus*, 2004.

[7] J. Gomez and C. Cachero. *OO-H Method: extending UML to model web interfaces*, pages 144–173. Idea Group Publishing, 2003.

[8] M. Hinz and Z. Fiala. Amacont: A system architecture for adaptive multimedia web applications. In *Workshop XML Technologien fuer das Semantic Web (XSW 2004)*, 2004.

[9] M. Hinz and Z. Fiala. Context modeling for device- and location aware mobile web applicationss. In *Workshop on Pervasive Mobile Interaction Devices (PERMID 2005) - Mobile Devices as Pervasive User Interfaces and Interaction Devices , Munich, Germany*, 2005.

[10] http://www.imagemagick.org. *ImageMagick project*, 2005.

[11] T. Isakowitz, M. Bieber, and F. Vitali. Web information systems - introduction. *Communications of the ACM*, 41(7):78–80, 1998.

[12] G. Klyne, F. Reynolds, C. Woodrow, H. Ohto, J. Hjelm, M. Butler, and L. Tran. *Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies*. W3C Working Draft, 2003.

[13] N. Koch, A. Kraus, and R. Hennicker. The authoring process of the uml-based web engineering approach. In *First International Workshop on Web-Oriented Software Technology*, 2001.

[14] B. Rutten, F. Frasincar, G. J. Houben, and R. Vdovjak. Hpg: a tool for presentation generation in wis. In *WWW (Alternate Track and Posters) 2004*, pages 242–243, 2004.

[15] R. Vdovjak, F. Frasincar, G. J. Houben, and P. Barna. Engineering semantic web information systems in Hera. *Journal of Web Engineering, Rinton Press*, 2(1&2):003–026, 2003.

[16] C. Ziegeler and M. Langham. *Cocoon: Building XML Applications*. New Riders, 2002.