Device Independent Web Applications – The Author Once – Display Everywhere Approach

Thomas Ziegert[1], Markus Lauff[1], and Lutz Heuser[2]

[1] SAP AG, Corporate Research, Vincenz-Prießnitz-Str. 1,
76131 Karlsruhe, Germany
{Thomas.Ziegert, Markus.Lauff }@sap.com
[2] SAP AG, Global Research & Innovation, Neurottstraße 16,
69190 Walldorf, Germany
Lutz.Heuser@sap.com
http://www.sap.com/research

Abstract. Building web applications for mobile and other non-desktop devices using established methods often requires a tremendous development effort. One of the major challenges is to find sound software engineering approaches enabling the cost efficient application development for multiple devices of varying technical characteristics. A new approach is to single author web content in a device independent markup language, which gets then adapted to meet the special characteristics of the accessing device. This paper describes our approach to single authoring, which was developed in the course a large European research project. The project has developed a device-independent language profile based on XHTML 2.0 and implemented a compliant rendering engine. We focus on layout and pagination capabilities of the RIML (Renderer Independent Markup Language) and show how authors can be assisted by development tools supporting device independent authoring.

1 Introduction

The World Wide Web has established itself as one of the most important sources for information as well as a perfect infrastructure for applications, which need to be accessible from anywhere. Web-enabled devices potentially offer access to globally adopted infrastructure. But why is this offer just potentially? Currently, most web content is optimised for the usage on a PC. This is still true despite the efforts of the Web Accessibility Initiative [1] which set standards for universal accessibility of web content. As more and more non-desktop devices enter the market, a convenient way to access web content and web applications using such devices is required. The industry addressed that requirement for a limited set of applications particularly in the B2E (Business-to-Employee) domain by developing device specific versions of such applications. This application-specific approach tends to be too costly and not manageable if scaled to a large number of diverse devices and applications. Therefore,

N. Koch, P. Fraternali, and M. Wirsing (Eds.): ICWE 2004, LNCS 3140, pp. 244–255, 2004.

more generic ways to prepare web content in a device-independent way are necessary.

Various approaches have been proposed to address the challenge of high quality applications at minimal costs. Some approaches (e.g. [2, 3]) automatically compile web content to fit on a target device. These approaches are based on HTML and use heuristics in addition to tag information to extract structure information, due to the fact that HTML is lacking the necessary semantics which is needed to perform the conversion to other markup languages. Therefore, some approaches replace HTML by another markup language which is semantically rich enough to serve as a basis for conversion [4, 5, 6]. However, none of these proposals is standards-based and they were therefore not widely adopted in the marketplace. To open the path for a widely adopted device-independent markup language for authoring web content, two considerable efforts have been launched recently: a) The W3C chartered the Device Independence Group to establish specifications supporting single authoring, b) a consortium of six European companies, named CONSENSUS[1] [7], has built a prototype which implements authoring and conversion tools for a Renderer-Independent Markup Language (RIML), which was also developed by this consortium. Both efforts cooperate intensively. The ultimate vision of device independence as stated in the charter of the W3C Device Independence working group [8] is to provide "Access to a Unified Web from Any Device in Any Context by Anyone". In this paper we focus on layout and pagination capabilities of the RIML (Renderer Independent Markup Language) and show how authors can be assisted by development tools supporting device independent authoring.

2 Requirements for Device Independency

In this section we briefly cover the most important requirements for a device independent markup language and explain how the Consensus project has dealt with those in the RIML. Following the author once, display everywhere approach, an application is written once and gets then adapted to a particular device, which is accessing it.

A perfect solution according to [9] should offer:

- a device independent markup language preserving the intent of the author,
- an adaptation system transforming it into a device specific form,
- means to retain the authors control over the final result/presentation.

Concentrating on the developer's part item 1 and 3 are the most important ones. To keep the intent of the author the markup language should offer semantically rich

[1] IST-Programme / KA4 / AL: IST-2001-4.3.2. The project CONSENSUS is supported by the European Community. This document does not represent the opinion of the European Community. It is also the sole responsibility of the author and not the responsibility of the European Community using any data that might appear therein.

markup elements. While XHTML 2.0 [10] provides certain means for device independent markup some necessary ingredients are missing. The RIML defines a language profile which is based on XHTML 2.0 and includes new elements for functionality, which was missing in current standards or proposals. In the reminder of this section we give a brief overview of these.

Due to the fact that devices widely vary in the amount of content their screens can accommodate there is a need for providing hints for content splitting and navigation between split up pieces of content. RIML therefore uses the `section` element of XHTML 2.0 as implicit hint for splitting and introduces new elements for combining layout with pagination and controlling the generation of navigation links between pages of split documents.

To provide means for the development of interactive web applications (business applications usually belong into this group) there is furthermore a need for elements dealing with form interaction. The XForms 1.0 [11] specification perfectly fulfills this requirement. The RIML language profile includes XForms 1.0, which strictly separates data from its presentation and keeps user interface elements device independent[2].

Apart from the fact that a device independent language allows for the generic description of a user interface for web applications, there should be means allowing for the inclusion of optional and alternative content. Even if this is somehow contradicting the author once approach, cause it allows for special versions of content for different devices, it enables the author to retain the control over the result of the adaptation process and allows him to consider the specialities of particular device classes (e. g. the support for certain audio or video formats). The RIML language profile therefore includes SMIL [12] Basic content control plus some extensions. Content control furthermore allows for the selection of device (class) specific style sheets. Style attributes like font sizes and weights tend to be very device specific, therefore RIML authors should provide different style sheets for different target markup languages and use the RIML content control mechanisms to select between them.

An optimal layout of a page or presentation unit on a certain device (class) heavily depends on the special characteristics of it. A generic layout specification, which then is adapted to the device accessing the content, is hardly to achieve. The layout should facilitate the usability of the particular application and therefore utilize the special characteristics of the device, e. g. the available screen size. Therefore RIML supports the authoring of different layouts for different device classes in a RIML document. Using content control the author is able to define when to use which layout.

[2] The latest draft of the XHTML 2.0 specification now also includes an XForms 1.0 module.

In this paper we focus on layout and pagination capabilities of the RIML (Renderer Independent Markup Language) and show how authors can be assisted by development tools supporting this novel features.

3 Layout and Pagination

Layout refers to the arranging pieces of content on a presentation unit PU[3]. The way layout is specified is a crucial problem when developing device-independent applications. Usually authors accomplish the layout by using frames and/or abusing tables for this purpose. Especially the latter is in conflict with the initial meaning of tables to serve as a structuring element, assembling multiple data records, each of which takes up exactly one row of the table. A generic layout specification, which then is adapted to the device accessing the content, is hardly to achieve. Automated layout generation, as described in [13, 14] might help for certain use cases, like assembling the layout of a remote control UI on several devices, but removes the author's control over the final result to a wide extent.

The layout should facilitate the usability of the particular application and therefore utilize the special characteristics of the device, e.g. the available screen size. Considering the limitation of the available screen size on certain devices another challenge of device independent authoring becomes apparent. Adapting to small screens requires the pagination (decomposition) of content, which in turn has to be taken into account, when specifying means for the layout of an authoring unit[4]. Usually some layout regions should be visible on all pages, e.g. menus and status bars; others include content, which should get split into a sequence of presentation units the pagination process generates.

Automated pagination support was a main design goal for RIML. Other approaches assume selectors that explicitly define device dependent breaks (like [15, 16]), in effect, falling back to device related authoring. In contrast, a RIML author requires a minimal knowledge of how a desired layout will be paginated by the RIML adaptation system. In this respect, RIML is related to other approaches [17, 18]. It differs from these in that it supports generic HTML like (row, column) constructs for adaptation, respectively applies adaptation to arbitrarily nested row and column structures.

---

[3] A presentation (PU) unit is this result of splitting the resource into a number of smaller units (PUs), which are presented to the user in a manner that is appropriate to the device (see DIWG2002]).

[4] An authoring unit is a piece of content the author is working with, but which is less than a document (see DIWG2002]).

Layout in RIML

Because of the reasons discussed in the last section RIML supports the specification of device (class) dependent layouts. Using Content Control the author is able to define when to use which layout. The overall layout of a RIML document is defined by using elements specified in the RIML layout module. The layout module defines a set of container types: rows, columns, grids, as well as frames. Whereas containers define the overall structure of a layout definition, frames are used to fill the several regions of the layout with content. Using different layouts and Content Control for layout selection allows content to be organized differently, depending on the target device and its unique properties. If a frame cannot simultaneously accommodate all of the content assigned to it in the target language, the content needs to be paginated. Pagination, the division of a RIML document into multiple pages, and navigation, the hyper linking among pages generated from a single authoring unit, are carried out automatically by the adaptation system.

RIML defines the following layout containers:

`grid`: A grid is a layout container allowing for the arrangement of container items in a grid layout. Container items can be (nested) other layout containers as well as frames. The parameter set of the grid element supports the specification of the maximum permissible number of columns in a grid and the direction into which grid items will be led out (either horizontally or vertically).

`column`: The column container is a special case of the grid container limiting the number of columns to one by definition.

`row`: The row container is a special case of the grid container forcing all contained items to be laid out in a single row.

`frame`: A frame is the only layout element content is assigned to, therefore a frame cannot contain any other layout elements. Each layout definition includes one or several frames. The assignment of content to a frame is done using the `riml:frameId` element.

Multiple frames can be used for the same presentation unit. Arranging frames differently in containers produces different layouts. RIML's Content Control gives the author a possibility for defining rules when to use which layout. While layout containers arrange the layout of frames in different ways, the actual content is assigned to frames only. All content in the body part of the document must be included into sections. Each section is assigned to a frame. The content of the section will be rendered within that frame (see Fig. 1). The content of a section will be ignored, whenever a section is not assigned to a frame or the frame does not exist. A RIML document should therefore define at least one frame in the document header. The overall layout of a document is defined by grouping multiple frames inside containers. RIML offers

different container types with differing layout and paginating behavior. All frames must be placed inside a container. The nesting of containers is allowed, i.e. a container can include other containers, whereas the innermost container must always include a frame. Therefore, a frame is the only type of container, which can be associated with content.

```
<head>
  <riml:layout eccdc:deviceClassOneOf="DeviceClass2">
    <riml:column riml:id="root-col">
      <riml:frame riml:id="head" />
      <riml:frame riml:id="nav-menu " />
      <riml:frame riml:id="home" />
    </riml:column>
</riml:layout>
</head>
<body>
    <section id="head-sec" riml:frameId="head">
      ...
    </section>
    <section id="menu-lsec" riml:frameId="nav-menu">
      ...
    </section>
      ...
</body>
```

Fig. 1. Assigning content to Frames

Pagination in RIML

XHTML 2.0 defines the block level element `section` as a means for structuring content. Apart from this a section defines an implicit page break, which is more "natural" than explicit page break markers. Therefore we decided to use sections as semantic hints for the adaptation system, when applying pagination. The author is therefore required to put all content, which should go onto the same screen (e. g. an input field and its label or a whole address form) into a section. Sections might be nested[5], whereas the innermost sections never get split. As every section moves into a certain frame defined by the layout in the head part of the RIML document, it is often the case, that certain regions of the layout should not get distributed over several pages, because they contain information, which should appear on every page, even if the document gets split. Taking Fig. 2 as an example, the LogoFrame as well as FrameB are defined as non-paginating frames (see also Fig. 3). For FrameA pagination was allowed (`paginate` is set to true).

[5] The current reference implementation does not support nested sections.

Fig. 2. Pagination and Layout Example

Therefore the content assigned to the LogoFrame and FrameB remains over the sequence of pages produced during adaptation. For FrameA the content gets distributed across all the pages the adaptation process produces. Fig. 3 shows a snippet of the markup, which was used to produce the results shown in Fig. 2. Section s10 contains special markup for guiding the automated generation of links between split pages. According to the definition used here, navigation elements should be generated for every page, showing a link to the previous and next page respectively. Navigation links are generated if and only if pagination occurs. The `navigation-links` element offers attributes allowing the author to define the type of link to be produced and to which Frame it is related. In Fig. 3 the scope attribute refers to FrameA, therefore navigation links are generated referring to pages containing sections from FrameA.

The size of a frame is determined by the adaptation system in order to accomplish pagination. However, the author should be able to provide the system with meta-information regarding the intended width of a frame. The pagination process considers the width of the virtual area on which the content gets finally rendered only (rendering surface), because this value can be reasonably defined. In case the browser supports no scrolling this value typically equals the physical screen width. Using the aforementioned frame elements the rendering surface can be divided into multiple parts. A frame serves as a bounding box in which the content will be laid out that it is assigned to. Using the following frame attributes the author is able to control the adaptation:

`minWidth` - the minimum width in pixels - the frame should never be smaller than this specified width.

`preferredWidth` - the recommended width of a frame is usually specified as a percentage in relation to the actual width of the rendering surface of the device. Absolute pixel values for preferredWidth are also supported.

```
<head>
  <title>Pagination and Layout</title>
  <riml:layout>
    <riml:column riml:id="col1">
      <riml:frame riml:id="LogoFrame" riml:paginate="false"
                  riml:minWidth="200"
riml:preferredWidth="400"/>
      <riml:frame riml:id="FrameA" riml:paginate="true"
                  riml:minWidth="200"
riml:preferredWidth="400"/>
      <riml:frame riml:id="FrameB" riml:paginate="false"
                  riml:minWidth="200"
riml:preferredWidth="400"/>
    </riml:column>
  </riml:layout>
</head>
<body>
  <section id="s1" riml:frameId="FrameA">
    ...
  </section>
  <section id="s2" riml:frameId="FrameA">
    ...
  </section>
  <section id="s10" riml:frameId="FrameB">
    <riml:navigation>
      <riml:navigation-links riml:scope="FrameA"
        riml:links="previous" riml:linksValue="relative-order"/>
      <riml:navigation-links riml:scope="FrameA"
        riml:links="next" riml:linksValue="relative-order"/>
    </riml:navigation>
  </section>
  <section id="s11" riml:frameId="LogoFrame">
    <strong>This is the sticky logo frame</strong>
  </section>
</body>
```

Fig. 3. Example Markup for Layout and Pagination

The `preferredWidth` attribute of a frame provides a hint to the adaptation process to determine the actual width of a frame. The author has to ensure that minWidth attributes in a RIML document layout can be obeyed for all devices. Therefore, the author should consider the device offering the smallest width of all targeted devices or better specify multiple layout definitions for different devices respectively device classes using Content Control.

Given those hints, a pagination algorithm has enough knowledge to paginate without exceeding the screen surface width. A similar approach cannot be applied with re-

spect to container height, due to mentioned undetectable user preferences. The determination of optimal height is based on two observations.
First, we expect that most browsers support vertical scrolling. Vertical scrolling was shown to be acceptable from a usability point of view [19], in contrast to two-dimensional panning. Vertical scrolling was shown to be disturbing, if certain limits are exceeded [20]. To avoid the latter, we enable the user to reduce (resp. increase) the size limit which is applied during pagination. In support of this, the adaptation system is to insert corresponding control hyperlinks. In effect, the user exploits the visible outcome of pagination to avoid undue vertical scrolling depths.

Besides Frames, the RIML furthermore defines other types of paginating elements, which are: paginatingGrid, paginatingRow and PaginatingColumn. With respect to pagination all of these container types operate on their child nodes, which might be either other containers or frames and which might be either paginating or not. To preserve a useable result, the pagination of nested layout elements is allowed for one branch only. Assuming that the hierarchy of layout elements forms a tree, this means that two paginating elements must not have a common ancestor.

For non-interactive content i.e. printed pages, the content will be split when the page is full and page numbers are inserted to provide a basic means of navigation. Specifications such as XML Print [17], CSS3 Paged Media [13], and XSL-FO [12] address that.

4 Tool Support

One major issue with new language proposals like RIML addressing Device Independence is the learning curve they require. The necessary authoring environment is often neglected in similar projects. Therefore, we decided to put considerable effort into the development of an authoring environment containing innovative tools, enabling the application developer to easily author in a device-independent way. Based on the Eclipse [21] open source platform, this authoring environment has been developed as a plugin, gathering a set of views and editors, synchronised around a common document model. The Consensus authoring environment provides a whole set of tools.

Besides an XML Editor consisting of a source text editor, code completion based on the RIML schemas, RIML language validation, and an XML tree editor, the toolset also provides tools, helping the developer to cope with the new concepts RIML introduce in a visual way. A Frames Layout View allows the author to get a first impression how the document looks like, based oh the abstract layout he defined in the document. The concept of this view is to show an early version of the frames layout of the current RIML document. Depending on the device class one chooses, the view will show the frames layout, including in each frame the names (or ids) of the section that belong to this frame. The XML text in the text editor is highlighted depending on

how the author places the focus in the Frames Layout View. Fig. 4 and 5 show the Frames Layout View for a PC and a smart phone, respectively.

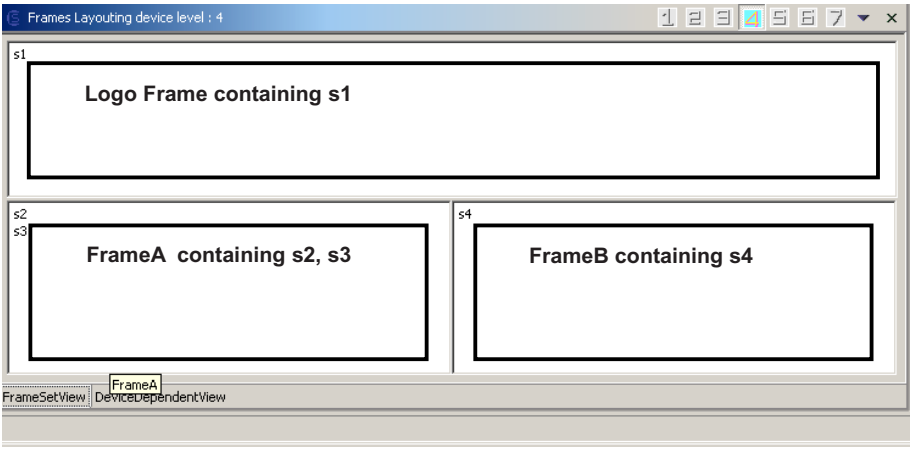

Fig. 4. Frames Layout View for Device Class 4

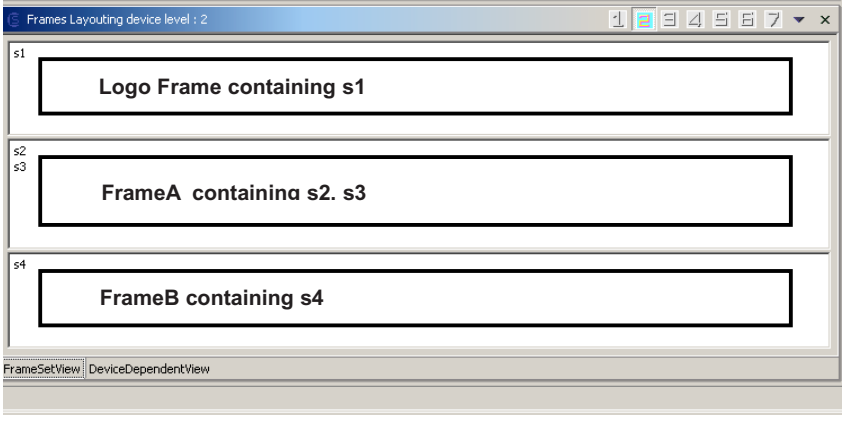

Fig. 5. Frames Layout View for Device Class 2

RIML authors often need to know how the developed document will be paginated depending on a device class. Therefore, the RIML Device Dependent View provides an overview of how the document is paginated, how many pages are created, and what they contain. The view was implemented similar to an XML tree view, presenting the split pages as a set of nodes. Fig. 6 shows an example of the device dependent view.

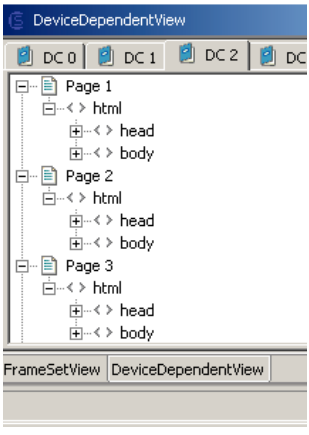

Fig. 6. Device Dependent View for Device Class 1

Apart from these rather abstract views, we have also integrated a set of available device emulators, allowing the author to see the actual result of the adaptation process on a particular device.

5 Conclusions

The paper has discussed the two particular aspects layout and pagination of content authoring for non-desktop devices and respective solutions developed in the Consensus research project. Rather than just specifying technology and markup to support single authoring, the Consensus project undertook the effort to implement a reference implementation and a set of tools supporting the author in applying these new concepts. Our experience with the Consensus prototype proved the feasibility of the concepts developed by the project. Test applications are now being built and will undergo a field test under close supervision of usability experts, ensuring that the developed concepts and technology are not just feasible, but also meet usability requirements. In parallel to that, the project works in close cooperation with the W3C to standardize key concepts explored in the project.

References

1. Web Accessibility Initiative, http://www.w3.org/WAI/
2. Bickmore, T.W.: Digestor: Device-independent Access to the World Wide Web, Proceedings of 6th International WWW Conference (1997)
3. Schilit, B.N., Trevor, J., Hilbert, D., Koh, T.K.: m-Links: An Infrastructure for Very Small Internet Devices. *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*. Rome, Italy, (2001) 122-131

4. Puerta, A., Eisenstein, J.: XIML: A Common Representation for Interaction Data, available at: http://www.ximl.org/documents/XIMLBasicPaperES.pdf
5. User Interface Markup Language, http://www.uiml.org
6. Eisenstein, J. et al.: Applying Model-Based Techniques to the Development of UIs for Mobile Computers, Proc. of the Conf. on Intelligent User Interfaces, Santa Fe, NM, USA, (2001)
7. Consensus Project Website, http://www.consensus-online.org
8. W3C's Device Independence Working Group, http://www.w3.org/2001/di/Group/
9. Butler, M., Giannetti, F., Gimson, R., Wiley, T.: Device Independence and the Web, IEEE Internet Computing, Sep./Oct. (2002) 81-86
10. McCarron, S., Axelsson, J., Epperson, B., Navarro, A., Pemberton, S. (eds): XHTML2, W3C Working Draft 5 August 2002, work in progress, http://www.w3.org/TR/xhtml2/
11. Dubinko, M., Klotz, L. L., Merrick, R., Raman, T. V.: XForms 1.0, W3C Recommendation 14. October 2003, http://www.w3.org/MarkUp/Forms/
12. Hoschka, P. (eds): Synchronized Multimedia Integration Language (SMIL) 1.0 Specification, http://www.w3.org/TR/1998/REC-smil-19980615 (1998)
13. Myers, B. A., Nichols, J.: Communication Ubiquity Enables Ubiquitous Control. 'Boaster' for Human-Computer Interaction Consortium (HCIC'2002). Winter Park, CO, Feb. (2002)
14. Nichols, J., Myers, B. A.: Automatically Generating Interfaces for Multi-Device Environments, Ubicomp 2003 Workshop on Multi-Device Interfaces for Ubiquitous Peripheral Interaction. Seattle, WA., October 12 (2003)
15. Adler, S. et al: Extensible Stylesheet Language (XSL), Version 1.0, http://www.w3.org/TR/xsl/
16. Lie, H. W., Bigelow, J. (eds): CSS3 Paged Media Module, work in progress, http://www.w3.org/TR/css3-page/
17. Mandyam, S. et al: User Interface Adptations, W3C Workshop on DI Authoring Techniques, http://www.w3.org/2002/07/DIAT
18. Keränen, H., Plomp, J.: Adaptive Runtime Layout of Hierarchical UI Components, Proceedings of the NordCHI 2002, Arhus, Denmark.
19. Giller, V. et al: Usability Evaluations for Multi-Device Application Development, Three Example studies, MobileHCI03, September (2003)
20. Baker, J. R.: "The Impact of Paging vs. Scrolling on Reading Passages", http://psychology.wichita.edu/surl/usabilitynews/51/paging_scrolling
21. The Eclipse Platform, http://www.eclipse.org/platform