Flexible pagination and layouting for device independent authoring

Gabriel Dermler
Michael Wasmund
IBM Lab Böblingen, Germany
{gabriel.dermler, mwasmund}
@de.ibm.com

Guido Grassel
Nokia Research Center
Helsinki, Finland
guido.grassel@nokia.com

Axel Spriestersbach
Thomas Ziegert
SAP Research
{axel.spriestersbach, thomas ziegert}
@sap.com

***Abstract*—We present the design of a markup language that is based on W3C standards and allows document authoring in a device independent fashion. We focus on layout and pagination capabilities enabled by the language and show that pagination is done in a highly structured manner and layout can be preserved to a high degree. We indicate constraints which are taken into account as well as information which cannot be assumed as available during pagination. We describe a user feedback loop to optimize pagination results based on user provided information.**

I. Introduction

A promise of mobile computing is that users can access remote applications using a great variety of devices with data processing capabilities including PCs, PDAs and cell phones.

While data transport is increasingly disappearing as an inhibiting factor for mobile access, there is still a lack of accepted models for user interaction and content presentation (UICP) which satisfy the needs of efficient application development and device specific characteristics. Various approaches are being considered stressing various constraints.

The mechanisms of the World Wide Web have set the standard for UICP based on PC browsers. The large number of existing HTML pages induced efforts for *automatic reauthoring* into representations which can be rendered on handheld devices with varying screen sizes. Such approaches (e.g. [1]) target transformations based on syntactical analysis, element specific reductions (e.g. image rescaling) or content exclusion based on assumptions regarding content semantics. Experiments have shown that, while generated content is legible, the aesthetical quality of presentations is low ([1]).

A second school of thought advocates the development of distinct UICP models for distinct devices. Providing a dedicated UICP model for every device ensures maximal UICP adaptation, at the expense of development cost ([2]). To alleviate this situation, approaches such as [2], [3] provide UICP models for a number different device classes. Still, UICP definition needs to be done several times.

Regarding implementation efficiency, implementing a UICP model once which can be be reused for every possible end user device is an ideal approach. To support *device independent authoring* various techniques have been considered:

- *abstract user interfaces* mapped onto a concrete UI representation of an employed device ([4], [5], [6])
- *content elision and transcoding* techniques (e.g.[1], [7])
- *presentation structure adaptations* such as for dynamic layouting and pagination (e.g. [7], [8]).

Our approach targets device independent authoring of documents which are similar to HTML pages. They are defined according to a novel *Renderer Independent Markup Language* (RIML). RIML is comparable to existing approaches, as it includes techniques of the types mentioned above. However, it differs from existing approaches for a number of reasons.

RIML stresses the separation of content definition (i.e. what is to be presented) from the description of dynamic adaptations which can be performed on the content in order to match varying capabilities of devices.

RIML is based on newest emerging standards. The current draft of XHTML2.0 ([9]) is used for content such as paragraphs, tables, images, hyperlinks, etc.. For form based interaction, XFORMS elements have been included [10].

Special *row* and *column* structures are used in RIML to specify content adaptation. Their semantics is enhanced to cover pagination and layout directives in case pagination needs to be done.

RIML eases device independent authoring in several ways. Authors familiar with XHTML / XFORMS can easily create content using RIML since it is based on such languages. Authors familiar with HTML tables or framesets can easily use row and column constructs for defining adaptation. To the best of our knowledge, RIML is the first approach addressing pagination for XForms.

Automated pagination support was a main design goal for RIML. Other approaches assume selectors that explicitly define device dependend breaks, 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 approaches [8], [7]. 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.

In this paper, we focus on RIML pagination capabilities. In Section 2, we present a basic example of pagination. In Section 3, basic adaptation mechanisms are discussed. In Section 4, enhancements are presented and in Section 5, row and column nesting is described for flexible pagination definition. In Section 6, the author and end user views are highlighted. We conclude with an outlook on future work.

II. Basic Pagination Elements

The basic principle for enabling RIML document pagination is that of a *paginating* row or column. To understand its purpose, consider as an example a simple document representing a list of news items:

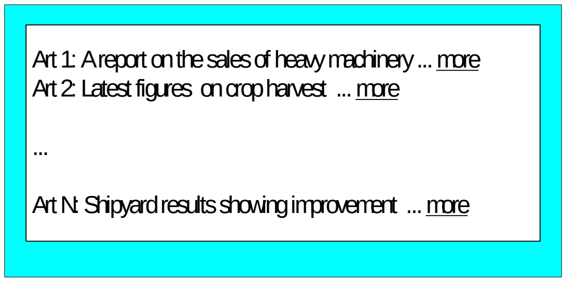
Art 1: A report on the sales of heavy machinery ... more
Art 2: Latest figures on crop harvest ... more
...
Art N: Shipyard results showing improvement ... more

Fig. 1. List of news without pagination

The resulting page corresponds to what the author of the document expects to be shown on a large screen where no adaptation has to be applied. Two cases can be distinguished when screen size is not sufficient to achieve this ideal result.

First, the width of the available screen surface may be such that the individual lines have to be wrapped around. However, the overall content can be still displayed on one page. Line wrapping is typically supported by today's browsers, so that no specific support is provided by RIML, to this end.

In the second case, the available screen surface is assumed to be too small to display the entire list at once. Pagination is applied resulting in several pages. Besides that, hyperlinks to switch between the pages are included (next, prev):

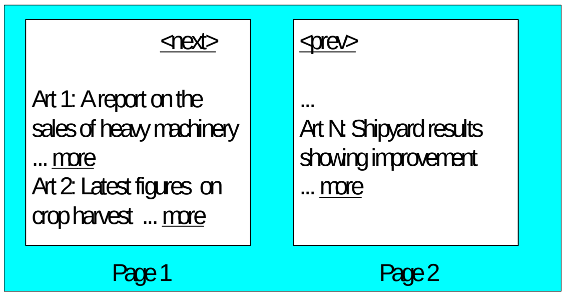
<next>
Art 1: A report on the sales of heavy machinery ... more
Art 2: Latest figures on crop harvest ... more
Page 1
<prev>
...
Art N: Shipyard results showing improvement ... more
Page 2

Fig. 2. List of news distributed on several pages

Pagination of the shown type is not part of today's browser functionality. In RIML, it is enabled by a *paginating column* or *row*. In the example, a paginating column is included (Fig. 3).[1]

The document contains as content XHTML elements denoting news titles and hyperlinks to retrieve the news. The RIML specification resembles what an XHTML document would consist of in its body part. To enable pagination, additional constructs are included:

1. *section* elements delimit the content which can be distributed onto different pages, i.e. all the elements appearing in a section must be shown on the same page. Sections therefore serve as a semantic hint for pagination. In the example, every article could be displayed on a different page.

2. a special section contains the *navigation elements* to be generated for every paginated page (prev, next). These are displayed only if pagination occurs and include attributes indicating the type of link to show (<prev>, <next>) as well as to which sections they are related. In the example, the *scope* attribute refers to the news sections container: navigation links are generated pointing to pages containing news sections which do not fit on the displayed page.

```
<html>
  <head>
    <riml:layout>
      <riml:column xml:id="c1"
           riml:minWidth="150px"
           riml:preferredWidth="300px">
        <riml:column xml:id="c1.1"/>
        <riml:paginatingColumn xml:id="c1.2"/>
      </riml:column>
    </riml:layout>
  </head>
  <body>
    <section riml:contId="c1.1" >
        <riml:navigation>
        <riml:navigation-links
          riml:scope="c1.2"
          riml:links="previous"
          riml:linksValue="relative-order" />
        <riml:navigation-links
          riml:scope="c1.2"
          riml:links="next"
          riml:linksValue="relative-order" />
      </riml:navigation>
    </section>
    <section riml:contId="c1.2" >
       Art1: A report on the sales of heavy
        machinery …
       <a href=http://www.comp.com/art1.riml>
          more </a>
    </section>
    . . .
    <section riml:contId="c1.2" >
       ArtN: Shipyard results showing
        improvement . . .
       <a href=http://www.comp.com/artN.riml>
          more </a>
    </section>
  </body>
</html>
```

Fig. 3. RIML document with a paginatingColumn

3. *containers* (here, a column) are associated with the sections of the document body by means of a "contId" attribute carried by the section. Column "c1.2" takes in the news sections, column "c1.1" takes in the section with navigation elements. Sections are inserted into containers in specified order, i.e. article 1 content comes before article 2 content, etc.

A container delimits a *rectangle area* on the screen where associated sections are presented. Therefore, it assembles sections whose content is semantically related, but might be distributed over several pages. Containers have *minimum and preferred values for their width*. The author uses the former to indicate the minimum width which makes sense for the container content. The latter indicates the width in the ideal case (i.e. when no adaptation is needed).

4. *Layout* specifies the *nesting of row and column containers*, similar to nested tables in HTML. In the example, the column "c1" includes columns "c1.1" and "c1.2", the former taking in the navigation section, the latter taking in the news sections. I.e. sections are organized in an

[1] For simplicity, we omit here the level of frames which are the RIML container type taking in RIML document sections.

outer column within which the navigation elements are displayed, below which the news related sections follow. The width for "c1" is applied to child columns, as they do lack own width values.

5. The "c1.2" column is specified as a *paginating container*, i.e. associated sections can be distributed over different pages. The actual distribution depends on how many sections can be displayed on the same page (see below). In the ideal case (i.e. sufficiently large screen), no distribution needs to take place.

III. Adaptation Mechanism Considerations

Translation to the markup language supported by a device concerns two aspects. First, *translation of section content* has to be done, for example, to HTML 3.2 or XHTML Mobile Profile in case of mobile devices or to WML for legacy mobile phones. How suitable a translation to VoiceXML can be remains to be seen.

Translation of layout representation is the second aspect to consider. Layout specification in RIML (via container width and nesting) has to be mapped onto elements of targeted markup languages. The width attribute in table elements is the equivalent to container width. XHTML MP provides, in a first approximation, tables similar to HTML. WML is more restrictive as it offers only a single column containing several rows.

For the example above, translations are straight-forward. The news section content consists of text and hyperlinks which have counterparts in HTML, XHTML MP or WML. The nested column layout is mapped to nested tables in HTML or XHTML MP. For WML, the outer column is mapped onto the one available column of WML while the sections contained in the inner columns are mapped onto WML rows.

Pagination is more complex. Put simply, the goal of pagination is to determine how much content fits onto a screen at once. Pagination also has to establish where within a RIML document, page boundaries are to be assumed. It is difficult to find an exact answer to these questions, due to limited knowledge about the constraints of content presentation.

Constraints such as screen size in pixels can be taken into account. User preferences (via browser settings) for screen resolution or fonts imply a low predictability of content appearance on the device screen. Hence, we do not assume that pagination can exactly derive the surface consumption implied by RIML documents. To control content appearance despite these uncertainties, we use two mechanisms.

First, authors specify minimum and preferred pixel widths for layout containers. Authors can base value selections on content appearance in their browsers using their browser settings. They can also use their experience as to which values lead to good content presentations. The pagination has to guarantee that a container width value is applied between the specified minimum and preferred width values. The closer the selected value to the preferred one, the better.

Given those metric hints, a pagination algorithm has enough knowledge to paginate without exceeding the screen surface width. A similar approach cannot be applied with respect 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 [13], in contrast to two-dimensional paning. Vertical scrolling was shown to be disturbing, if certain limits are exceeded [11]. 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.

IV. Repeating Content During Pagination

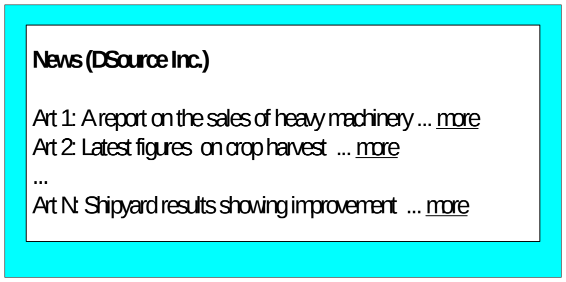
News (DSource Inc.)
Art 1: A report on the sales of heavy machinery ... more
Art 2: Latest figures on crop harvest ... more
...
Art N: Shipyard results showing improvement ... more

Fig. 4. Example page including a title

Pagination should not simply cut the content of a RIML document into distinct pieces. Reconsider the earlier example and assume that an identifier of the news provider is included (Fig. 4). In case of pagination, the news provider is expected to be repeated on all resulting pages:

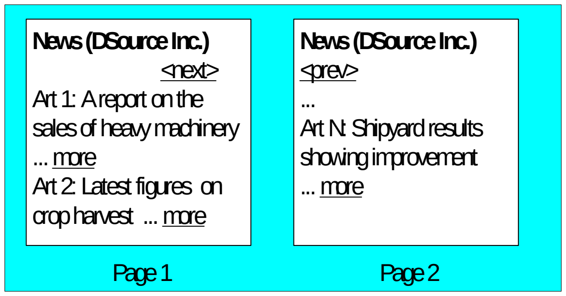
News (DSource Inc.)
<next>
Art 1: A report on the sales of heavy machinery ... more
Art 2: Latest figures on crop harvest ... more
Page 1
News (DSource Inc.)
<prev>
...
Art N: Shipyard results showing improvement ... more
Page 2

Fig. 5. Repeated content on paginated pages

To support this, RIML enhances the semantics of *non paginating containers*. For the example, an additional section (containing the news provider identifier) and a column placing this section in the layout are included:

```
. . .
    <riml:layout>
         . . .
        <riml:column xml:id=”c1.0” />
        . . .
    </riml:layout>
    . . .
    <section riml:contId=”c0.0” >
      News (DSource Inc.)
    </section>
. . .
```

Fig. 6. Inclusion of a news provider section

A non-paginating container implies that child elements are repeated on resulting pages (which include parts of the container content). In the example, containers “c1.0” (i.e. the title section) and “c1.1” (i.e. the navigation section) are to be repeated on all resulting pages. Repetition is also required with respect to “c1.2” (containing the news sections). As this container is paginating, repetition implies a distribution of its content over the resulting pages to achieve consistency across the paginated pages.

V. Layouting via Container Nesting

RIML permits to nest rows and columns, enabling authors to create arbitrarily structured layouts similar to well-known techniques used with HTML.

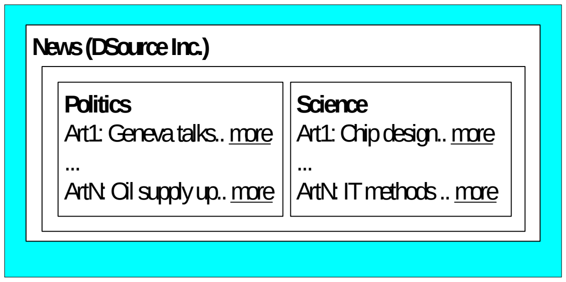

Fig. 7. Example with nested column and row structures

For pagination, nesting needs deeper consideration. Assume that above page has to be displayed on a narrow screen where just one news container (politics or science) can be fitted in. To enable pagination between the two parts, we include a paginating row containing the containers for politics (“c1.r1.c1”) and science (“c1.r1.c2”) (see Fig. 8).

A number of implications arise from this. First, the usage of a row container implies a preferred width value, which is to be applied to the elements contained (i.e. the “politics” and “science” parts). This allows for taking into account desired relative widths for the individual parts.

The pagination effect is twofold. First, as the row container is of paginating type, its children can be distributed over different pages (Fig. 9). Such pagination would, for instance, occur for narrow device screens which have sufficient height to display a long list of items. For device screens with limited screen height, a second pagination scheme is enabled (Fig. 10).

Insertion of navigation links is adaptive. If pagination occurs only within the row container, links allowing to switch between the news categories are inserted. In case pagination is done both within row and column containers, navigation links for both levels are inserted. This corresponds to the specification in the document where one navigation element has a scope set to the row container, while a second element has a scope set to the “politics” container.

In consequence, nesting of row and paginating containers allows a high degree of flexibility in layout specification, similar to the layout techniques used for today’s PC browsers. The pagination semantics defined for the nesting allows performing the implied adaptation in case of constrained device screens. As shown in Fig. 9 and 10, adaptation is applied only on the necessary level. Moreover, navigation is enabled according to the applied pagination scheme.

```
<html><head>
<riml:layout>
  <riml:column xml:id="c1"
   riml:minWidth=”150px”
   riml:preferredWidth=”600px”>
   <riml:column xml:id=”c1.0” />
   <riml:paginatingRow xml:id=”c1.r1”>
     <riml:column xml:id=”c1.r1.c1”
       riml:preferredWidth=”300px” >
       <riml:column xml:id=”c1.r1.c1.1”/>
       <riml:paginatingColumn
        xml:id=”c1.r1.c1.2” />
     </riml:column>
    <riml:column xml:id=”c1.r1.c2”
      riml:preferredWidth=”300px”>
      <riml:column xml:id=”c1.r1.c2.1”/>
      <riml:paginatingColumn
       xml:id=”c1.r1.c2.2” />
    </riml:column>
   <riml:paginatingRow />
</riml:layout>
</head>
<body>
  <section riml:contId=”c1.0” >
    News (DSource Inc.)
  </section>
  <section riml:contId=”c1.r1.c1.1” >
    <riml:navigation>
      <riml:navigation-links
        riml:scope=”c1.r1”
        riml:links=”previous”
        riml:linksValue=”relative-order”/>
      <riml:navigation-links
        riml:scope=”c1.r1”
        riml:links=”next”
        riml:linksValue=”relative-order”/>
      </riml:navigation>
    <riml:navigation>
      <riml:navigation-links
         riml:scope=”c1.r1.c1.2”
         riml:links=”previous”
      riml:linksValue=”relative-order” />
      <riml:navigation-links
        riml:scope=”c1.r1.c1.2”
        riml:links=”next”
        riml:linksValue=”relative-order”/>
    </riml:navigation>
  </section>
  <section riml:contId=”c1.r1.c1.1” >
    Politics
  </section>
  <section riml:contId=”c1.r1.c1.2” >
    Art1:Geneva talks …
  </section>
  <section riml:contId=”c1.r1.c1.2” >
    ArtN:Oil supplies up …
  </section>
    <!-- science part skipped -->
</body></html>
```

Fig. 8. RIML document with nested containers

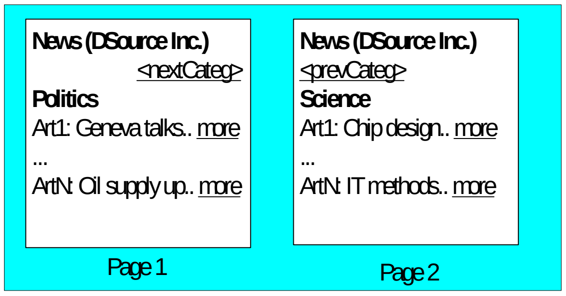

Fig. 9. Pagination within the row container

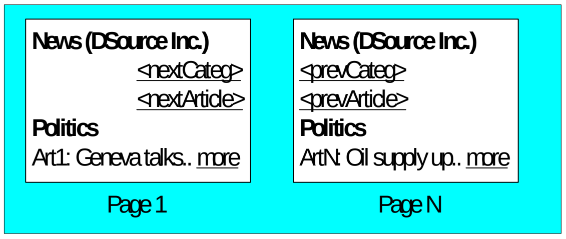

Fig. 10. Simultaneous pagination in rows and columns

VI. Authoring and End User Views

A main design goal for RIML was to simplify authoring with respect to device knowledge. The presented layout structuring is largely independent of device characteristics. Container definitions and nesting is fully device independent. Width indications are content related, i.e. device independent. An author specifies the preferred width as the equivalent he would like to see for a container when presented on an unconstrained screen. The minimal value is the one he assumes to make sense for the container content.

The structuring of content into sections which can be displayed atomically on the smallest targeted device is the only aspect where a RIML author has to be aware of device specifics. More exactly, he has to be aware of the minimal available screen surface to be supported. The current RIML specification corresponds to the capabilities of the pagination we have initially designed and are implementing.

The adaptation concepts presented here ensure that a user is presented, as close as possible, with the preferred layout defined by an author, i.e. RIML has built-in layout preservation capabilities. If a screen is large enough, the preferred layout will be displayed. In case, adaptation is needed, it is constrained to the highest level possible in the layout specification of the document. Layout below that level is preserved. If pagination occurs, adaptation is done in a highly structured manner. Navigation links are inserted according to the layout nesting structure which is adapted during pagination.

The adaptation mechanisms ensure that paginated pages do not violate screen width limits. In contrast, we expect that RIML adaptation implies vertical scrolling for the end user. In addition, a user control is assumed allowing a user to reduce content limits applied during pagination. This user feedback loop allows coming close to what a user considers acceptable vertical scrolling, on an individual basis.

VII. Outlook

We have completed the first specification of RIML and the design of the adaptation system needed to transform RIML into HTML, WML and XHTML MP markup. We are implementing this system including required pagination algorithms and, in particular, support for paginating XFORMS. In parallel, RIML extensions are considered to support additional features. Both current and future designs of RIML are kept such that other techniques (e.g. image rescaling) can be included into RIML. RIML design, although not focusing on inventing such mechanisms, is kept open to this end.

The work described in this paper is being done within the EU supported project CONSENSUS (IST-Program / KA4 / AL: IST-2001-4.3.2)[2] with participation of industrial partners. More information about Consensus can be retrieved from its Web site [12].

References

[1] T. W. Bickmore, B. N. Schilit, “Digestor: Device-independent Access to WWW” Proc. of the 6th WWW Conf., Santa Clara, CA, USA 1997.
[2] Unwired Planet Inc., Dev Guide V1.0, Redwood Shores, USA, 1997 .
[3] I. Cooper, R. Shufflebotham, “PDA Web Browsers: Implementation Issues”, Tech. Report, Canterbury Comp. Lab, Univ. of Kent, UK, 1995.
[4] “User Interface Markup Language”, www.uiml.org ,
[5] “Extensible Interface Markup Language”, www.ximl.org .
[6] J. Eisenstein 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] S. Mandyam, et.al. , “User Interface Adptations”, W3C Workshop on DI Authoring Techniques, http://www.w3.org/2002/07/DIAT .
[8] H. Keränen, J. Plomp , “Adaptive Runtime Layout of Hierarchical UI Components”, Proceedings of the NordCHI 2002, Arhus, Denmark.
[9] XHTML2, http://www.w3.org/TR/xhtml2/ .
[10] XFORMS, http://www.w3.org/MarkUp/Forms/ .
[11] J. R. Baker, “The Impact of Paging vs. Scrolling on Reading Passages”, http://psychology.wichita.edu/surl/usabilitynews/51/paging_scrolling
[12] Consensus web site, http://www.consensus-online.org .
[13] V. Giller et.al. „„Usability evaluations of multi-device applications. example studies”, submitted to the Conf. on Mobile HCI, Sep 2003,

[2] This document does not represent the opinion of the European Community. It is also the responsibility of the authors and not the responsibility of the EC using any data that might appear therein.