XML Document Adaptation Queries (XDAQ): An Approach to Adaptation Reasoning Using Web Query Languages

Michael Kraus[1,2], François Bry[1] and Kazuhiro Kitagawa[2]

[1] Institute of Computer Science, University of Munich
[2] Graduate School of Media and Governance, Keio University
(michael.kraus, francois.bry)@informatik.uni-muenchen.de

Abstract. Adaptive web applications combine data retrieval on the web with reasoning so as to generate context dependent contents. The data is retrieved either as content or as context specifications. Content data is, for example, fragments of a textbook or e-commerce catalogue, whereas context data is, for example, a user model or a device profile. Current adaptive web applications are often implemented using ad hoc and heterogeneous techniques. This paper describes a novel approach called "XML Document Adaptation Queries (XDAQ)" requiring less heterogeneous software components. The approach is based on using a web query language for data retrieval (content as well as context) and on a novel generic formalism to express adaptation. The approach is generic in the sense that it is applicable with all web query and transformation languages, for example with XQuery and XSLT.

1 Introduction

Adaptive web applications combine data retrieval on the web with reasoning so as to generate context dependent contents. Data is retrieved either as content (for example fragments of a textbook or e-commerce catalogue), or as context specifications (for example a user model or a device profile). For example, the presentation of learning material for students (content) may be adapted to the student's estimated knowledge about the topics (context) [16]. Current adaptive web applications [8, 6] are often implemented using ad hoc and heterogeneous techniques.
This paper describes a novel approach called "XML Document Adaptation Queries (XDAQ)" requiring less heterogeneous software components. The approach is based on using a web query language for (content as well as context) data retrieval and on a (novel) generic formalism to express adaptation. The approach is generic in the sense that it is applicable with all web query and transformation languages, for example with XQuery and XSLT.
The benefits and contributions of XDAQ are as follows:

– The goal is not to implement a single or several adaptive web applications, but to provide a framework that can be used to implement a wide variety

of adaptive web applications, ranging from educational hypermedia to e-commerce systems and device-independent web applications for desktop and mobile computers. Sample applications, however, will be implemented to test and improve the XDAQ approach.
- XDAQ does not rely on a single, fixed query language. XDAQ can be used with a wide variety of web query languages, among other with XQuery and XSLT. XDAQ can also be applied with more than one query language simultaneously.
- XDAQ is a generic formalism for expressing adaptation of arbitrary XML documents such as XML data and databases, HTML documents, XSLT transformations, XSL documents, or XLink linkbases.
- Non-intrusive adaptation rules separate between web contents and adaptation rules. This greatly improves modularity of adaptive web application modeling and helps to avoid unwanted redundancy.
- XDAQ defines a formal model of how to process XML documents together with adaptation rules, third-party links and style rules, thus avoiding existing ambiguities when combining these approaches.

There are also two extensions which further broaden the application spectrum of XDAQ. The first extension allows XDAQ adaptation rules not only to be used with XML documents, but also with non-XML data like CSS style sheets, XPath expressions, or URIs. The approach is based on the idea of separating data model and syntax of those languages, and providing a data model based on XML while keeping the original syntax.
The second extension is a processing model which allows processing of adaptation rules to be distributed between client, server and proxy. The distribution is determined by the user's privacy settings, and device characteristics like network bandwidth and processing power. This approach allows XDAQ to be used for both typical client-side adaptations like in educational hypermedia [8], as well as typical server-side adaptations like in tourist information systems [6]. Furthermore, this approach enables a single adaptive web application to be used both in desktop environments (fast network, powerful clients) and mobile environments (slow network, slim clients). For simplicity, these two extensions are not described in the present paper.
The rest of this paper is organized as follows: The next four sections introduce step by step the concepts of XDAQ. Section 6 then discusses how the XDAQ framework can be used to model complex adaptive web applications. Finally, section 7 briefly presents an XDAQ prototype and future research issues.

2 Modeling Context Information

Adaptation is based on *context information*. XDAQ assumes context information to be stored in one or more *context documents*. These documents are referenced by URIs (see section 3), therefore they can be located not only on the client

or the server of the adaptive web application, but anywhere on the web. Context documents can be modeled using XML, RDF, or any other web modeling language.
The following is an example of a (freely shaped) XML context document which will be used by the examples in the rest of the paper:

```
<context>
  <knowledge topic="html">beginner</knowledge>    <!-- User Data -->
  <language>en</language>
  <language>ja</language>
  <interest field="music">Pop</interest>
  <device type="large"/>                          <!-- Device Data -->
  <device pixel_width="1024"/>
  <device pixel_height="768"/>
  <time value="12:56"/>                           <!-- Environment Data -->
  <date>30 June 2003</date>
  <temperature value="24C"/>
  <location value="Yokohama"/>
  <situation>meeting</situation>
  <history>                                       <!-- Browsing History -->
    <page uri="http://xyz.com/index.html">
      <page uri="http://xyz.com/page1.html">
        <page uri="http://xyz.com/page2.html"/>
      </page>
      <page uri="http://xyz.com/page3.html"/>
    </page>
  </history>
</context>
```

Context information considered in this paper covers the following (the list does not preclude further context information that is not explicitly mentioned in this paper to be used with the XDAQ approach):

- User: knowledge about certain topics (beginner about HTML), language competence (speaks English and Japanese), or interests (pop music).
- Device: type ("large" device), resolution (1024x768 pixels).
- Environment: date and time, temperature, location, situation (meeting).
- Browsing history

The term *context information*, as it is used in this paper, is an abstraction of notions like user profile and device profile, which are used in other approaches for adaptive (web) applications [2, 17]. Context information subsumes these notions, and even goes beyond them: Any data found on the web can be used as context information, not only data about the user or his device.
Context documents are typically administered by the adaptive web application itself, but this could also be done by the web browser or by the operating system. Maintaining and updating context information can be done in three ways. Most information, like device data, environment data, the browsing history and part of the user data can be updated automatically. Information like the *situation* can be explicitly set by the user. Data like test results and the user's knowledge or interest on certain topics can be updated by the adaptive web application. The user's interests could also be deduced from the browsing history, for example.
The definition of suitable standard vocabularies describing context information for certain domains has been identified as one of the most urgent tasks within the

web context adaptation community [17, 11]. There is a common need for a standardized and simple vocabulary which ensures interoperability, and which allows for proprietary extensions for domain-specific or application-specific context information. In spite of this, it is a major goal of XDAQ to provide adaptation methods that are independent from the actual vocabulary of the context information, and thus can be used with a wide variety of adaptive web applications. XDAQ makes no further assumption about context information than that it is represented in a language like XML or RDF. While the examples in this paper only use XML as language to represent context information, the XDAQ approach can also be used with the existing CC/PP or UAProf [18] vocabularies, which are based on RDF.

Context information may be modeled flat (like the *user*, *device*, and *environment* sections), or structured (like the *browsing history* section). Many traditional adaptive hypermedia systems [8, 16], as well as the CC/PP framework [13], which is targeted at device independence, simply use flat lists of property/value pairs for representing context information. In contrast to this, XDAQ also allows to model complex structured context information. In conjunction with XDAQ's approach of modeling adaptation based on web query languages, this enables adaptation not only to rely on the actual content of the context information, but also on structural information. For example, adaptation could rely on certain browsing patterns of the user, which can be queried from the structure (not the contents) of the browsing history.

An advantage of allowing for freely shaped contexts is that the roles of context and content can be exchanged, if the application requires it. Consider for example the browsing history information from above. This information can be used by an educational hypermedia application to make recommendations to the user about the suitability of yet unvisited pages, like it is done in [8]. In this case, the browsing history serves as context information. Another application, for example the history function of a web browser, might use the same information but present it directly to the user. In this case, the browsing history serves as content. Not requiring a fixed shape or vocabulary for context information makes it possible to change the angle from which information is looked at, context or content.

3 Expressing Adaptation with Query Languages

With XDAQ, adaptation is expressed through *adaptation rules* of the form:

```
<rule>
  <href>URI</href>
  <if context="URI" type="language">query</if>
  <copy context="URI" type="language">query</copy>
</rule>
```

There are two kinds of adaptation rules: *if-rules* and *copy-rules*. These rules each consist of a set of *target node bindings* and a set of *queries*. The target node bindings attach an adaptation rule to a set of nodes of one or more XML documents, called the *target nodes* of that rule. The documents containing the target nodes are called *target documents* of that rule. The queries each consist of a reference to a context document, called the *context document* of that query, and an expression of a web query language like XPath, XSLT, or XQuery. It is possible to use more than one query language within a set of adaptation rules and even within a single rule. This section discusses the queries of an adaptation rule, and the following section discusses the target node binding.

An if-rule consists of one or more if-queries. The semantics is as follows: Each query is evaluated against its context document, which is given by the `context` attribute. If the result of at least one query is empty, that is, if there aren't any nodes in the context document that match the query, then each target node of the rule is deleted from its containing document. Put the other way round, the target nodes of an if-rule are remain in their containing document if and only if the results of all the if-queries are not empty. The nodes are deleted together with their all their children and descendants.

The following example illustrates the use of if-rules in conjunction with an HTML document. As the queries do not specify their context document, the default of the application is used, for example the context document maintained by a web browser.

```
<rule>
  <href>#paragraph1</href>
  <if type="xpath">//knowledge[@topic='html'][.='beginner']</if>
</rule>

<rule>
  <href>#paragraph2</href>
  <if type="xpath">//knowledge[@topic='html'][.='expert']</if>
</rule>

<html>
  ...
  <p>This page gives an explanation of HTML.</p>
  <p id="paragraph1">Very <em>simple</em> introduction to HTML, in
  addition to the following standard explanation.</p>
  <p><em>Standard</em> explanation of HTML.</p>
  <p id="paragraph2">Additional <em>advanced</em> aspects of HTML.</p>
  ...
</html>
```

The two rules can be used to adapt the HTML document to the user's knowledge about HTML:

- By default, the document contains a "standard explanation" of HTML. If the user is neither classified as *beginner* nor as *expert*, then only the standard explanation will be displayed.
- If the user is classified as *beginner* about the topic *HTML*, then an additional "simple introduction" to HTML will be displayed before the standard explanation.

– If the user is classified as *expert*, then additional "advanced aspects" of HTML will be displayed after the standard explanation.

Actually, the HTML document contains both the standard explanation as well as the simple introduction and the advanced aspects. According to the context information, however, either the simple introduction or the advanced aspects or both are deleted from the document by an XDAQ processor before it is displayed by the browser.
A copy-rule consists of one copy-query and zero or more if-queries. The semantics is as follows: The copy-query is evaluated against its context document, given by the `context` attribute. Then the children of each of the rule's target nodes are replaced by the result of the query. Thus, if the rule has n target nodes, then n copies of the query result are inserted into the target documents of that rule. If the rule also contains if-queries, then first the if-queries are evaluated against their context documents. Only if the results of all if-queries are not empty (similar as with if-rules), then the copy-query is evaluated, otherwise the copy-rule has no effect.
The following example illustrates the use of a copy-rule in conjunction with an XML document and an XSLT transformation:

```
<rule>
  <href>#xpointer(//recommendation)</href>
  <if type="xpath">//interest[@field='music']</if>
  <copy type="xpath">//interest[@field='music']/text()</copy>
</rule>

<library>
  <recommendation>Rock</recommendation>
  <album title="Radio Musicola" artist="Nik Kershaw" genre="Pop"/>
  <album title="Calling All Stations" artist="Genesis" genre="Rock"/>
  <album title="Script From A Jester's Tear" artist="Marillion" genre="Progressive Rock"/>
  <album title="The Works" artist="Nik Kershaw" genre="Pop"/>
</library>

<xsl:template match="library">
<html>
  ...
  <p>Recommended Albums:</p>
  <ul><xsl:apply-templates select="album[@genre=//recommendation]"/></ul>
  <p>Other Albums:</p>
  <ul><xsl:apply-templates select="album[@genre!=//recommendation]"/></ul>
  ...
</html>
</xsl:template>
```

The copy-rule is used to adapt the XML document by copying the user's preferred music genre into the document. The result of the copy-query is only copied if the result of the if-query is not empty, that is, if the context document specifies the user's preferred music genre. Without the if-query, the contents of the rule's target node would be deleted (set to empty) if the context document did not specify such information. The XSLT transformation generates an HTML document containing two lists: The first is a list of *recommended* albums, based on the user's interest that has been copied from the context document. The second list contains all other albums.

Obviously, instead of XPath, any other query and transformation language, for example XQuery, could be used. Thus, XDAQ does not rely on a single, fixed query language. The only, obvious requirement is that the query language can be used with the query's context document, for example, using an RDF query language for CC/PP documents.

4 Binding Adaptation Rules to Document Nodes

This section describes how adaptation rules are attached to nodes in an XML document. The approach of XDAQ is non-intrusive, that is, adaptation rules can be attached to nodes in a document without having to change to document itself. This is important because it makes it possible to non-proprietary build an adaptive system referring to contents collected from the web.
Non-intrusively binding is done by using references that point from an *adaptation pool* document, that is, a document containing a set of adaptation rules, into the rules' target documents, for example the HTML and XML documents from the examples in the previous section. A single adaptation rule may have more than one target node as well as more than one target document, that is, the target nodes may be contained in more than one XML document. For example:

```
<rule>
  <href>http://xyz.com/page1.html#xpointer(//p[@class='beginner'])</href>
  <href>http://xyz.com/page2.html#paragraph11</href>
  <href>http://xyz.com/page2.html#paragraph12</href>
  <if type="xpath">//knowledge[@topic='html'][.='beginner']</if>
</rule>
```

This if-rule is attached to a class of `p` elements in document `page1.html`, as well as to two single elements in document `page2.html`.
As reference mechanism URIs together with XPointer fragment identifiers are used. XPointer allows not only to select arbitrary nodes from an XML document, but also a set of locations, that is, simply speaking, substrings of the content of a node [12]. Thus it is possible to model not only coarse adaptation like deletion of elements, but also very fine-grained adaptation using attributes and even substrings of nodes as targets. A single XPointer fragment identifier may select more than one location, as in the example above.
Because the adaptation rules are separated from the documents that they affect, both documents can be created and maintained by different persons and be stored in different locations, even on different servers. In a large adaptive web application which is created and maintained by a number of people, the task of adaptation modeling can be performed simultaneously to other tasks. The goal is that, once the overall structure of an XML document (HTML document, XSLT style sheet and so on) is fixed, it should be possible for one person to fill the document with data, while another person may work on the adaptation rules, without interfering each other. It is even possible for a third party to define

adaptation rules for a set of documents without having write access to those documents.
Note that the same approach, that is, pointing from an external document into the main document, is also used by CSS and XSLT style sheets as well as XLink linkbases. Defining adaptation rules non-intrusively, both improves modularity and avoids unwanted redundancy in adaptive web applications.
In contrast, an intrusive approach, that is, an approach that is attaching the adaptation rules directly to nodes in an XML document, introduces redundancy, as adaptation rules queries have to be copied for each target node. With an intrusive approach, the example from above may look like the following (note that adaptation queries in real applications might be much more complex than those used in the examples in this paper):

```
<p class="beginner" xdaq:if="//knowledge[@topic='html'][.='beginner']">...</p>
<p class="beginner" xdaq:if="//knowledge[@topic='html'][.='beginner']">...</p>
<p class="expert">...</p>

<p id="paragraph10">...</p>
<p id="paragraph11" xdaq:if="//knowledge[@topic='html'][.='beginner']">...</p>
<p id="paragraph12" xdaq:if="//knowledge[@topic='html'][.='beginner']">...</p>
```

Even if the queries are not directly inserted into the document, but only pointers to them, the document has still to be altered if it is to be equipped with adaptation, thus hindering modularity:

```
<p class="beginner" xdaq:href="http://adapt.org/adaptation-pool1.xml#rule1">...</p>
<p class="beginner" xdaq:href="http://adapt.org/adaptation-pool1.xml#rule1">...</p>
<p class="expert">...</p>

<p id="paragraph10">...</p>
<p id="paragraph11" xdaq:href="http://adapt.org/adaptation-pool1.xml#rule1">...</p>
<p id="paragraph12" xdaq:href="http://adapt.org/adaptation-pool1.xml#rule1">...</p>
```

Furthermore, using an intrusive approach as above, it is not possible to attach an adaptation rule to attributes or substrings of the contents of an element, because XML does not allow to let attributes have attributes themselves, for example. Thus, in contrast to XPointer fragment identifiers, only coarse adaptation, based on full elements, can be modeled.
A system using intrusive adaptation rules like above is [7]. Also CSS media queries [15] use an intrusive approach by attaching *media queries* (adaptation queries) directly to CSS document nodes. Therefore both of these approaches suffer from unwanted redundancy and a lack of modularity.

Absolute vs. Relative Pointers

XDAQ allows the use of both absolute and relative URIs [1] for the target node bindings of an adaptation rule. This allows to define adaptation rules for either a specific document or for a class of documents.
The example at the beginning of this section uses absolute URIs as target node bindings, while the examples in section 3 all use relative URIs.
Relative URIs are relative to the document that is processed by an XDAQ processor, called the *main document*. For example, if a set of adaptation rules is used

to adapt document A, then all relative URIs used as document node bindings for adaptation rules are resolved using the URI of document A as base URI. If the same rules are used to adapt another document B, then the URI of document B is used as base URI.
The right use of absolute and relative URIs allows a single adaptation pool document, that is, a single set of adaptation rules, to be used for a whole website, even if some pages use different adaptation rules that all the others. Relative URIs are used to describe the default adaptation, that is used by almost all pages of the web site, while absolute URIs are used to describe exceptions from that, for example to use different adaptation rules for the title page.
Note that while XLink also allows the use of both absolute and relative URIs [9], there is no absolute referencing in CSS or XSLT style sheets. Therefore, if a single page of a web site, is meant to be styled different than all the other pages, either a different style sheet has to be used for that page, or the nodes in that page have to be identified by id and class attributes that are not used elsewhere on the web site.

5 Adaptive Web Document Processing

In an adaptive web application, adaptation rules exist together with links and style rules, which both can change the structure and contents of the main document. This may lead to ambiguous situations like this:

- A link inserts an image after element `e1`
- An adaptation rule deletes element `e1` from the document

There are two possible semantics for this set of rules: Either the image is inserted after `e1` and then `e1` is deleted, keeping the image. Or first `e1` is deleted, then there is nothing that is referenced by the link, resulting in the image not being inserted. Adding style rules, things can get even more complicated. To avoid such ambiguities, XDAQ defines a formal processing model for adaptive web documents. A *web document* is referenced by a URI and consists of the following parts:

- The *main document*, which is the XML document actually referenced by the URI of the web document. The main document might be the result of an XSLT transformation, a XQuery query of a database and so on. This step, however, lies out of the scope of the XDAQ processing model.
- A set of *processing information* documents, that is, XDAQ adaptation pool documents, XLink linkbases and CSS style sheets. These documents are either referenced by the main document, recursively referenced by other processing information documents that already have been referenced, or they are known to an XDAQ processor by any other means.
- Links from the main document to processing information documents are inline (intrusive) links. They may also contain inline (intrusive) adaptation

rules, in order to prevent unnecessary transfers of unused documents. Otherwise, the main document contains no other intrusive links, style rules or adaptation rules.

The XDAQ processing model considers adaptation rules, links and certain style rules as transformations of the main document. Such style rules include CSS pseudo-selectors like `:before`, or the `display` property with a value of `none`. Properties that affect fonts, colors and similar are not considered to be transformations, of course.
An XDAQ processor, for example contained in a web browser, performs the following steps in order to process an adaptive web document:

1. Parse the main document m.
2. Recursively parse all processing information documents linked by m, after evaluating each link's adaptation rule. This leads to a set of adaptation pool documents $\mathcal{A}$, a set of linkbases $\mathcal{L}$, and a set of style sheets $\mathcal{S}$. Accordingly, there exist sets of adaptation rules $\mathcal{R}(\mathcal{A})$, links $\mathcal{R}(\mathcal{L})$, and style rules $\mathcal{R}(\mathcal{S})$.
3. Process $\mathcal{R}_{\mathcal{A}}(\mathcal{A}) = \{r \in \mathcal{R}(\mathcal{A}) \mid T \cap \mathcal{A} \neq \emptyset, T = target\_documents(r)\}$. In other words, process all adaptation rules which have adaptation pool documents as target nodes. The rule r is not processed as a whole, but only those target nodes contained in adaptation pool documents. This step ensures that all adaptation rules, which are subject to adaptation, are adapted themselves before they are applied to their target nodes in the next step (see section 6).
4. Process $\mathcal{R}_{\mathcal{L}}(\mathcal{A}) = \{r \in \mathcal{R}(\mathcal{A}) \mid T \cap \mathcal{L} \neq \emptyset, T = target\_documents(r)\}$ and $\mathcal{R}_{\mathcal{S}}(\mathcal{A}) = \{r \in \mathcal{R}(\mathcal{A}) \mid T \cap \mathcal{S} \neq \emptyset, T = target\_documents(r)\}$. This step applies all remaining adaptation rules except those affecting the main document, as the main document can still be changed by links and style rules in the next step.
5. Process $\mathcal{R}(\mathcal{L})$ and $\mathcal{R}(\mathcal{S})$.
6. Process $\mathcal{R}_m(\mathcal{A}) = \{r \in \mathcal{R}(\mathcal{A}) \mid T \ni m, T = target\_documents(r)\}$, resulting in m', the adapted main document. This step finally applies all remaining adaptation rules to the main document, after is has been modified by links and style rules, thus giving adaptation rules the lowest priority in the adaptation process.

This processing model avoids ambiguities like the one described at the beginning of this section. The lack of a formal processing model for XLink is considered the main reason why major web browsers still do not support more than XLink simple links, despite being the XLink specification a recommendation for more than two years [10]. Several implementations have reported problems when combining XLink with other mechanisms that alter the XML document structure, for example XSLT transformations [14]. The combined processing model for XLink and CSS described in [19], however, is much more complicated than the XDAQ processing model and still lacks the concept of adaptation.

6 Adaptation Reasoning

After the previous sections introduced the technical aspects of XDAQ (modeling context information, definition of adaptation rules, binding of adaptation rules to document nodes and the outline of a formal processing model), this section takes a look at *adaptation reasoning*, that is, how adaptive applications can be modeled using the XDAQ framework. By applying if-rules and copy-rules to certain nodes of web pages, databases, or transformations, various adaptation technologies like adaptive presentation and adaptive navigation support [2] can be modeled. Two aspects of adaptation are important here: adaptation *actions*, for example, deletion of a node from a document, and the query part of adaptation rules.

The kind of adaptation that can be modeled using if-rules is very similar to the approach of *Author's Views* described in [4]. With Author's Views, certain parts of an XML document (for example paragraphs in an HTML document) are labelled with identifiers. A *view* corresponding to one of those identifiers then deletes all parts of the document not labelled with that identifier. Author's Views can be implemented using the `display` property of CSS with a value of `none`, and the user can choose between different views by selecting different style sheets. The concept of if-rules is based on that approach, but extends the simple notion of identifiers as labels to queries over a set of context documents, thus allowing to implement *conditional text* [2]. This means that the *views* are selected automatically based on the context information.

The metaphor of *variables* in programming languages can be used to illustrate adaptation modeling with copy-rules. As in the example from section 3, the target node of a copy-rule can be considered a variable, whose value is set by a copy-rule. If the copy-rule also contains if-queries, then the original content of the target node can be considered the default value of the variable. If the conditions expressed by the if-queries do not apply, the variable keeps its default value.

If-rules used to model Author's View-like adaptation in conjunction with copy-rules providing variables, are considered to be able to cover a large variety of adaptive web applications, namely technical documents like educational hypermedia or manuals. Such documents can be adapted to *beginner* users and *expert* users, or to *small* devices and *large* devices, as illustrated by the examples in the previous sections. However, adaptation may consist of more than just deleting nodes, for example:

- inserting nodes into a document
- replacing node contents
- ordering
- splitting and merging of documents
- disabling of hyperlinks

In order to provide such functionality, one could introduce new types of rules, one for each functionality: *insert-rule*, *replace-rule*, and so on. This, however, would

yield a complete XML transformation language equipped with adaptation facilities, for example an XSLT/XQuery-like language with if/copy/insert/replace-rules.
The approach of XDAQ is different: take an existing, non-adaptive language like XLink, and attach adaptation rules from the outside (non-intrusive). The result is a two-step approach in many cases: XLink models for example insertion of nodes into the main document, and an attached if-rule models the adaptation:

```
<link id="link1" xlink:type="extended">
 <locator xlink:type="locator" xlink:href="a.xml#paragraph11-short"/>
 <locator xlink:type="locator" xlink:href="b.xml#paragraph11-long"/>
 <arc xlink:type="arc" xlink:actuate="onLoad" xlink:show="embed"/>
</link>

<rule>
  <href>linkbase.xml#link1</href>
  <if type="xpath">//device[@type='large']</if>
</rule>
```

The link inserts a long version of *paragraph11* from document `b.xml` into document `a.xml`, at the place of the short version of *paragraph11*. The if-rule, however, deletes the link if the device is not "large". This has the effect that the long version is only inserted on large devices, whereas on other devices, the short version of the paragraph is used as it is.
XDAQ adaptation rules can be used to directly adapt HTML documents, or to adapt XML data, which is then transformed into an HTML document, for example, or to adapt XML transformations, linkbases, style sheets, and any combination of these. For example, by applying adaptation rules to hyperlinks (as opposed to the actual content of a web page), adaptive navigation methods like hyperlink sorting, hiding, and annotation [2] can be modeled. Finally, XDAQ allows adaptation rules to be applied to other adaptation rules, which could be called *second-order* adaptation. A detailed investigation of these issues within the scope of sample applications remains to be done in the future (see section 7).
Adaptation may not only rely on "typical" context information, which is stored in context documents, but also on properties of the main document. For example:

> If the document is displayed on a small device, then cut of all long paragraphs (more than 200 characters) after 200 characters and add "..." at the end.

Something like this can be modeled using an XSLT transformation, for example, which checks the condition based on the main document (paragraph longer than 200 characters) and performs the adaptation action (cut off paragraph and add "..."). The condition based on context information is modeled by an attached if-query. Thus again a two-step approach is used: conditions based on the main document are modeled using a non-adaptive transformation mechanism, while conditions based on context information is attached via adaptation rules.
The same example could also be implemented using CSS, with an appropriate if-rule attached to the following set of style rules:

```
p[‘‘length>200’’] { ‘‘cut: 200’’ }
p[‘‘length>200’’]:after { content: "..." }
```

The current CSS specification is both unable to express the notion of the length of a paragraph as well as cutting off text. These two features, however, could easily be introduced, as well as other, similar functionalities desirable for adaptive web applications like ordering, splitting and merging of documents, and hyperspace disabling.
The question whether such functionalities are considered as transformations in the sense of XSLT and similar transformation or query languages, or simply as style properties in the sense of CSS, cannot be answered unambiguously. Both approaches are applicable, but the differences are as follows:

- A general-purpose transformation language like XSLT can be used to implement any desired functionality, whereas a style sheet language like CSS has to be extended each time a new functionality is added.
- Using a transformation language, authors have to write an algorithm, that is, *how* the adaptation action is to be performed. In the above example, this could be done using either a loop or recursive templates together with the `strlen` function of XPath. With CSS, authors only have to define the result of the adaptation action, that is, *what* should be performed. The algorithm itself is part of the CSS processors and has to be implemented only once (and not by the author of the web application).
- Even if the CSS approach can only provide a limited set of functionalities, a web application can combine it with a general-purpose transformation language. Functionalities not covered by CSS can still be implemented using the transformation language.

The idea of considering certain kinds of adaptation actions as style/layout instead of transformations is described in more detail in [3]. Future research will further investigate this issue and combine XDAQ with an extended version of CSS, that is able to perform adaptation actions like splitting and merging documents.

Query Languages for Adaptation

When used to express adaptation, query languages face different requirements than when used within other applications.
In addition to simply retrieve values from context documents (content queries), for example screen resolution, it must also be possible to express complex structural queries. Using structural queries, certain kinds of browsing behaviour can be detected from the history data in the context document, for example. According to the browsing behaviour, users can be classified into groups like beginner or expert, which is important for educational hypermedia systems.
Educational hypermedia typically also makes use of various stochastic functions. For example, adaptation might be based on the standard deviation of a series of

test results. Query languages used for adaptation should provide a wide range of such functions.
Another feature widely used in adaptive hypermedia systems is ranking of list items. Ranking can be performed on both contents, like a list of news headlines, or on navigation (links), like a list of keyword search results. It is sufficient for the query language to generate rank values, for example by adding a rank attribute to list elements. The ordering itself is considered as a style/layout issue, which can be expressed by a CSS rule like this:

```
ul.recommendation li { sort-key: @rank; order: descending }
```

Furthermore, query languages used for adaptation must be able to perform reasoning based on ontologies. For example, it should be possible to model something like $small \subset medium \subset large$, that is, everything that can be displayed on a small device can also be displayed on a medium sized device, and that can be displayed on a large device. Also, ontologies can be used to provide interoperability between different context vocabularies.

7 XDAQ Prototype and Future Research

A prototype implementing the XDAQ framework has been developed in Java. At present, it exists only as a stand-alone tool, but it is being integrated into a web browser, a web server and a proxy, thus implementing the extended processing model described in the introduction. The prototype includes processing of adaptation rules as defined in sections 3 and 4, the processing model of section 5, as well as processing of non-XML data (CSS) as described in the introduction.
Future research will concentrate on the issues discussed in section 6. Especially the combination of XDAQ with the web query language Xcerpt [5] is promising, as Xcerpt is meant to be extended with functionalities needed for adaptation reasoning. Also, further investigation of the difference between adaptation considered as *transformation* and adaptation considered as *style/layout* is on the way. At present, the development of two adaptive web applications is planned to investigate these topics and the issues discussed in section 6, adaptive student learning material and an access-anywhere organizer.

8 Acknowledgements

The authors are thankful to Nobuo Saito and Keita Matsuyama, Keio University, for useful suggestions, and to Epson and DAAD for financial support.

References

1. T. Berners-Lee et al. Uniform Resource Identifiers (URI): Generic Syntax. RFC 2396, August 1998. http://www.ietf.org/rfc/rfc2396.txt.

2. P. Brusilovsky. Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 6(2-3):87–129, 1996. http://www.contrib.andrew.cmu.edu/∼plb/UMUAI.ps.
3. F. Bry and M. Kraus. Adaptive Hypermedia made simple using HTML/XML Style Sheet Selectors. In *Proceedings of the 2nd International Conference on Adaptive Hypermedia and Adaptive Web Based Systems (AH 2002)*, number 2347 in LNCS, pages 472–475. Springer, May 2002. http://www.pms.informatik.uni-muenchen.de/publikationen/#PMS-FB-2002-1.
4. F. Bry and M. Kraus. Advanced Modeling and Browsing of Technical Documents. In *Proceedings of the 17th ACM Symposium on Applied Computing (SAC 2002)*, pages 520–524. ACM Press, March 2002. http://www.pms.informatik.uni-muenchen.de/publikationen/#PMS-FB-2001-11.
5. F. Bry and S. Schaffert. A Gentle introduction into Xcerpt, a Rule-based Query and Transformation Language for XML. In *International Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, June 2002. http://www.pms.informatik.uni-muenchen.de/publikationen/#PMS-FB-2002-11.
6. K. Cheverst, K. Mitchell, and N. Davies. The role of adaptive hypermedia in a context-aware tourist guide. *Communications of the ACM*, 45(5):47–51, 2002.
7. The Consensus Project. http://www.consensus-online.org/.
8. P. De Bra et al. AHA! The Next Generation. In *ACM Conference on Hypertext and Hypermedia*, May 2002. http://wwwis.win.tue.nl/∼debra/ht02.pdf.
9. S. DeRose et al. XML Linking Language (XLink) Version 1.0. W3C Recommendation, June 2001. http://www.w3.org/TR/xlink/.
10. B. DuCharme. XLink: Who Cares?, March 2002. http://www.xml.com/pub/a/2002/03/13/xlink.html.
11. R. Gimson. Using Delivery Context to Achieve Device Independence: Outstanding Issues. In *W3C Delivery Context Workshop*, March 2002. http://www.w3.org/2002/02/DIWS/submission/gimson-issues.html.
12. P. Grosso et al. Xpointer framework. W3C Working Draft, July 2002. http://www.w3.org/TR/2002/WD-xptr-framework-20020710/.
13. G. Klyne et al. Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies. W3C Working Draft, March 2003. http://www.w3.org/TR/2003/WD-CCPP-struct-vocab-20030325/.
14. M. Kraus. A Toolkit for Advanced XML Browsing Functionalities. Diploma thesis, Institute for Computer Science, University of Munich, November 2000. http://www.pms.informatik.uni-muenchen.de/publikationen/#DA_Michael.Kraus.
15. H. W. Lie et al. Media queries. W3C Candidate Recommendation, July 2002. http://www.w3.org/TR/2002/CR-css3-mediaqueries-20020708.
16. W. Nejdl. Adaptivity in the KBS Hyperbook System. In *2nd Workshop on Adaptive Systems and User Modeling on the WWW*, 1999. http://www.kbs.uni-hannover.de/∼henze/paperadaptivity/Henze.html.
17. L. Suryanarayana and J. Hjelm. Profiles for the Situated Web. In *The Eleventh International World Wide Web Conference (WWW2002)*, 2002. http://www2002.org/CDROM/refereed/214/index.html.
18. Wireless Application Group User Agent Profile Specification, November 1999. http://www1.wapforum.org/tech/documents/SPEC-UAProf-19991110.pdf.
19. N. Walsh. XML Linking and Style. W3C Note, June 2001. http://www.w3.org/TR/xml-link-style/.