Ersch. in : World Wide Web ; 15 (2012), 4. - S. 447-481

XCML: providing context-aware language extensions for the specification of multi-device web applications

Michael Nebeling · Michael Grossniklaus · Stefania Leone · Moira C. Norrie

Abstract There is a vast body of research dealing with the development of context-aware web applications that can adapt to different user, platform and device contexts. However, the range and growing diversity of new devices poses two significant problems to existing approaches. First, many techniques require a number of additional design processes and modelling steps before applications can be adapted. Second, the new generation of platforms and technologies underlying these devices as well as upcoming web standards HTML5 and CSS3 have partly changed the way in which web applications are implemented nowadays and often limit the way in which they can be adapted. In this paper, we present XCML as one example of a domain-specific language that tightly integrates context-aware concepts and adaptivity mechanisms to support developers in the specification and implementation of multi-channel web applications. In contrast to most existing approaches, the objective is to use a more lightweight approach to adaptation that can dynamically evolve and support new requirements as they emerge. Our solution builds on versioning principles in combination with a context matching process based on a declaration of context-dependent variants of content, navigation and presentation in terms of context expressions at different levels of granularity that are specific to the application. To support this, a formally defined context algebra is used to parse and resolve the

M. Nebeling · S. Leone · M. C. Norrie
Institute of Information Systems, ETH Zurich, 8092 Zurich, Switzerland

M. Nebeling
e-mail: nebeling@inf.ethz.ch

S. Leone
e-mail: leone@inf.ethz.ch

M. C. Norrie
e-mail: norrie@inf.ethz.ch

M. Grossniklaus (✉)
Computer Science Department, Portland State University, Portland, OR 97201, USA
e-mail: michagro@cecs.pdx.edu

context expressions at compile-time and to determine the best-matching variants with respect to the client context at run-time. We present the language concepts and a possible execution environment together with context-aware developer tools for the authoring and testing of adaptive features and behaviour. We also report on two case studies: the first shows how our general approach allows for integration with existing technologies to leverage advanced context-aware mechanisms in applications developed using other platforms and languages and the second how existing web interfaces can be systematically extended to support new adaptation scenarios.

Keywords multi-channel support in web applications • domain-specific language approach • integration with modern web platforms and technologies

1 Introduction

Web developers currently have to deal with the increased proliferation and diversity of new devices that vary widely, not only in terms of screen size and resolution, but also in the different input and output modalities they support. It is becoming increasingly difficult for developers to cater for these new adaptation scenarios in a responsive manner. While the majority of web sites still use a static interface that covers only a certain range of devices, others have started to provide separate versions tailored to specific browsing clients. For example, in response to the new generation of smartphones and tablet computers, many popular sites nowadays offer special mobile editions optimised for the smaller screen space and touch interaction. Looking at the other end of the spectrum, there is also a recent trend towards much larger forms of high-resolution screens and interactive surfaces. However, most application developers, including web site providers, have not even started to consider these despite the fact that they are now becoming commonplace in offices, meeting rooms, public spaces and homes.

The need to support multi-channel access in web applications and the challenges associated with this are not new. The adaptation of web applications to different use contexts has been a popular topic in web engineering. Considerable efforts have been made to uniformly address the requirements of context-awareness and adaptivity in web applications. In research, however, the focus has often been on different modelling techniques with the aim of providing comprehensive approaches that allow applications to be designed, not only for multiple devices and different modalities, but also with support for personalisation, internationalisation and location-awareness. Many popular methods involve a number of different models of the user interface, often using several abstraction levels and logical descriptions, to then be able to transform these and generate the concrete and final interfaces adapted for different client contexts. Most approaches promoted by research aim to be powerful in that they try to cater for as many aspects of context-awareness as possible. To achieve this, they typically involve a number of additional design processes, such as task and domain modelling, which require more abstract descriptions of the user interface. However, in the web design practice, this is not practical for many web developers that typically tend to start with various prototypes of the concrete user interfaces.

In addition, it is nowadays often required to build on a range of different software development kits to cater for specific device models, but the code generation techniques often lag behind and do not allow for easy integration with the latest platforms and technologies. On the other hand, there is also a newer generation of platforms and frameworks, e.g. Silverlight, JavaFX and OpenLaszlo, that have been specifically designed to support multiple run-time environments. The supported target platforms often range from native web and mobile applications to desktop solutions that run outside the browser. Moreover, upcoming web standards HTML5 and CSS3 increasingly play a role since they offer new features for client-side adaptation and therefore new approaches. For example, HTML5 is increasingly used as a basis for many new multi-device frameworks such as PhoneGap[1] or Joshfire,[2] and CSS3 media queries provide a way of specifying alternative designs with respect to different device and media features. Many of these new technological approaches carry the potential of providing more lightweight and elegant solutions to supporting multiple viewing and interaction contexts, but, in comparison with some of the model-based approaches, only address some aspects of context-awareness. In particular, the underlying languages often lack common concepts and vocabulary, let alone a unified method, for the adaptation to different contexts.

We therefore decided to revisit the topic of supporting context-awareness in web applications and started to investigate ways of aligning existing conceptual and technological solutions. Our overall goal is to support developers in the design and specification of adaptive web applications using more lightweight approaches that can evolve dynamically as new adaptation requirements emerge. As a first step in this direction, we propose XCML, the *Extensible Context-Aware Markup Language*, that tightly integrates context-aware concepts and adaptivity mechanisms into XML. The main benefit of building XCML on top of XML is that it provides a basis for integration with many other frameworks and platforms that are also based on XML technologies, and can therefore bridge the gap to more specialised solutions. However, we need to emphasise that XCML is only one example of a domain-specific language that implements the principles presented in this paper. The intention is therefore not to replace existing user interface languages and technologies with a new language, but to leverage and extend these by applying the proposed techniques. One of the key criteria that distinguishes our approach from others is that we build on versioning principles and a context matching approach based on finer granularity of the context representation. This allows applications based on XCML to be extended more dynamically without the need to modify existing adaptivity features. While we promote the use of XCML to support multi-channel access, the same principles used to support this kind of adaptivity can also be applied to address the more common kinds of context-awareness that developers have to consider these days. This means that next to the adaptation of content, structure and presentation of web applications for different device characteristics in terms of screen settings and input modalities, which are the focus of this paper, the support for adaptivity based on XCML also includes, for example, localisation and internationalisation, as well as other kinds of personalisation.

In addition to the core features of the language also described in [27], this article provides three important contributions: First, we give a formal definition of the

[1]http://www.phonegap.com/

[2]http://joshfire.com/

context-aware concepts behind the language as a basis for implementations on top of other languages and technologies. Second, we present a possible execution environment and also address the need for tool support for developers by showing how the XCML development environment can be leveraged to facilitate, not only the authoring, but also the often cumbersome debugging and testing of context-aware applications, a requirement that has not been addressed so far. Finally, we demonstrate that our general approach can integrate with existing platforms and applications and cater for different adaptation scenarios using the examples of two case studies.

We begin in Section 2 by discussing related work and reviewing existing approaches. Special attention will be given to context-aware markup languages and data models that form the basis of our approach which is then presented in Section 3. Section 4 explains the execution environment for our language and the processing steps involved based on an example. This is followed by a brief summary of the tool support in Section 5 before moving on to present the two case studies as part of the evaluation in Section 6. Finally, we discuss the results of these studies in Section 7 and give concluding remarks in Section 8.

2 Background

The requirement for web sites to be able to adapt to context is well documented in the literature [23]. In general, context-aware web sites define a set of adaptation operations that are executed in response to client requests in order to adapt the content, navigation and presentation according to the client context. There are many different approaches to achieve such an adaptive behaviour in applications. We will start our review by looking at the different methods and frameworks promoted by research. We will then take a closer look at the state-of-the-art in terms of implementation platforms and emerging technologies. Special attention will be given to the underlying languages and models as a basis for comparisons between the approaches and with XCML.

As already mentioned, many popular web design methodologies have been extended with support for context-aware concepts and mechanisms. For example, WebML [9] has introduced new primitives for modelling the behaviour of adaptive web sites [8]. The models used in WebML support both user-independent, context-triggered adaptivity actions as a form of active context-awareness [7] as well as user-behaviour-aware actions [6]. The latter introduces a rule-based adaptation technique along the Event-Condition-Action (ECA) paradigm, which also forms the basis of the Bellerofonte framework [11]. In Bellerofonte, adaptivity rules are separated from application execution by means of a separate ECA server that processes events and enacts the active logic [10]. This decoupled runtime management of adaptivity features allows developers to dynamically extend the adaptive behaviour of an application and is also something we want to support with XCML and the underlying implementation platform.

In UWE [21], adaptation is based on the Munich Reference Model [25] using UML's Object Constraint Language. UWE has also formed the basis of an aspect-oriented approach that promotes the use of special classes to achieve a systematic separation of general system functionality and context adaptation. For example, the work presented in [1] uses the notion of aspects to specify common types of adaptive navigation, e.g. to support adaptive link reordering, annotating and hiding in response to user behaviour. While these kinds of user adaptation scenarios are not in the focus of this paper, we will show that the concepts of XCML are of a general nature and can in principle also be used for such user-dependent aspects of context-awareness.

The specification of adaptation is also an integral part of the Hera methodology [22]. In Hera, web site elements can be annotated with appearance conditions that control how the hypertext presentation is adapted [17]. Hera can be used in combination with different models to configure a run-time environment such as the Hera Presentation Generator (HPG) [16] or AMACONT [15]. While HPG is tailored to Hera, AMACONT is a more general implementation platform that supports adaptation along document, content unit and media components defined in a layered component-based XML format [14]. The focus of the language is on component-based development as a means to support reuse. It also supports the specification of adaptive properties by attaching metadata to components using a form of switch-statement to distinguish different contexts. However, adaptation rules are defined in an *if-then-else* manner using conditional statements, which can become increasingly difficult with larger context models. It was therefore our intention to develop a different concept based on a formally defined context algebra and matching process in order to distinguish different contextual states and to perform the associated adaptation operations.

More recently, Hera and AMACONT have been extended with support for a combination of rule-based and aspect-oriented approaches [5, 31] to enable the specification of multiple high-level design concerns. To this end, HyperAdapt [30] is ongoing work that currently investigates how to detect aspect interactions, conflict resolution and controlled linking of web adaptation aspects since this can become a major concern in complex applications based on this model [31]. In XCML, we instead build on a best-match approach that does not require the full specification of rules and conditions for the controlled linking between them. We will show that the enabling concept of context matching expressions, not only adds to the scalability of applications and support for new adaptation scenarios, but also aids design simplicity for developers.

In a second stream, research on user interface description languages has spawned a number of different models, frameworks and tools, with the most prominent example being the CAMELEON reference framework [4]. CAMELEON separates out different levels of user interface abstraction to support the adaptation to user, platform and environment contexts. The suggested development processes typically unfold along a four-step, top-down approach, starting with domain concepts and task modelling, followed by subsequent transformation steps from abstract to concrete and the final user interfaces. The authoring of adaptive and multi-modal user interfaces has been the subject of extensive research, e.g. [33]. A diversity of languages have been proposed, such as UIML [20], UsiXML [26] and MariaXML [34], most of which follow the CAMELEON model, but provide different device and modality-independent methods to specify the appearance of the user interface and interaction with the application logic. By contrast, XCML does not define a particular context or user interface model and is instead designed to work in combination with existing models and languages.

The two approaches that mainly inspired our work on XCML are [12] and [18]. Both of them present general and extensible architectures for context-aware data access and management independent of a particular design methodology or implementation platform. The approach in [12] is based on the General Profile Model that again uses a rule-based matching process, but rather than building on the typical ECA techniques, it compares client rules to adaptation rules by means of a parametrised profile, a condition and a parametrised configuration [13]. A similar approach is presented in [18] with the difference that an object-oriented version model is used to declare multiple variants of structure, content, view and layout with respect to different client contexts. The approach builds on formally defined scoring and weighting functions and a matching algorithm to determine the best-matching version of the web site according to the current context [19]. The work presented in this paper builds from these principles since we believe that the underlying concepts are particularly suitable for addressing new adaptation requirements in a responsive manner. XCML provides the same separation of concerns and the versioning concepts in the form of a domain-specific language on top of XML. We chose XML as the basis because it is the language used in the majority of approaches and therefore provides starting points for integration with existing frameworks and platforms.

Early research on Intensional HTML (IHTML) [36] already demonstrated how context-aware concepts could be integrated into markup languages. IHTML can be used for declaring web content in different versions inside the same HTML document. The concepts used in IHTML were later extended to form the basis of Multi-dimensional XML (MXML) [35] as a general format for the definition of semi-structured data in multiple versions. However, these early languages still suffered from two major design issues. First, the multi-dimensional concepts were not tightly integrated with the markup language and instead needed to be specified in separate blocks of proprietary syntax. This renders existing HTML/XML validator and related developer tools void as an attempt to parse IHTML/MXML documents will find syntax errors and therefore fail. Moreover, it becomes extremely difficult to manage such versioned documents, at least for a considerable number of dimensions, if the possible states are not clearly defined in some sort of context model.

Contrasting this extensive body of research with state-of-the-art development practices shows a significant gap between the methodical approaches at the conceptual level and existing web development platforms at the technological level. For example, research has promoted the importance of a number of design processes such as task modelling [4]. In practice, however, these steps are often skipped and developers instead produce directly the concrete user interfaces by building on state-of-the-art web development platforms such as Silverlight, JavaFX and OpenLaszlo. This is reasonable since the majority of them have been specifically designed to support multiple run-time environments. On the other hand, the underlying XML-based languages, which are respectively XAML, FXML and LZX, use different and often very limited notions of context. With XCML, it is therefore our goal to develop language extensions that leverage advanced context-aware concepts and multiple adaptivity dimensions while fostering a tighter integration with existing languages and technologies.

Despite the need to support context-awareness and adaptivity in web applications, it is only recently that some of these ideas have made it to the web standards.

For example, HTML5[3] provides new markup elements that allow for annotation of document structure and for semantic content which can inform the adaptation process. Moreover, it provides ways for embedding multimedia and graphical content into the DOM which allows also for adaptations of this type of content. Finally, HTML5 leverages advanced features through extensions of the DOM interface that enable native web applications to support client-side storage and therefore offline usage, as well as supporting location-awareness with the help of geographical location information. Also CSS3 provides new concepts such as media queries[4] for the definition of stylesheets with respect to different media types, e.g. for on-screen reading or printing. Media queries thereby provide access to, not only the media type, but also media-specific features. By querying these properties, certain styles can be included or excluded depending on factors such as the size of the browser viewport or client screen, the resolution of the output device, its orientation or aspect ratio. Equally important is the fact that media queries are re-evaluated every time a relevant variable changes. This means that they are sensitive to client-side events and provide support for adaptivity. For example, using media queries, the layout of a web site can be adapted automatically or completely replaced when, for example, the device orientation changes or the size of the browser window exceeds a certain threshold. As mentioned earlier, our work on XCML was therefore also motivated by the increasing support for adaptivity in the latest web standards and aimed at exploring their level of support as well as the limitations with respect to different adaptation requirements.

3 XCML

In this section, we introduce the concepts of XCML and show how the proposed features support developers in the specification of context-aware and adaptive web applications. Our presentation is structured around the following three core concepts of XCML.

Application-specific context model Context-aware applications based on XCML must first define the context dimensions and possible contextual states in a context model specific to the application. We provide a formal definition of the context representation used in XCML and show how applications can gradually extend the context model to address new requirements as they emerge.

Context-aware component model Following the common separation of concerns between content, navigation and presentation, XCML uses components to represent content and structure independent of presentation instructions, while separate layout elements provide the presentation templates. We show how the versioning principles behind XCML allow for context-dependent variants at each of the different conceptual levels.

[3]http://www.w3.org/TR/html5/

[4]http://www.w3.org/TR/css3-mediaqueries/

Context expression language and algebra XCML uses a general matching process to support context-aware adaptation. This matching process is based on context expressions that are used both to annotate web site components and to express client contexts. We present the features of the expression language and explain XCML's context algebra used for the evaluation of context expressions.

Most modern context-aware platforms start from a well-defined application that acts as a default or as a fallback in the absence of context information. This default application is then adapted using context information in order to improve or augment system behaviour. A distinguishing feature of any approach is therefore how developers express this adaptation. As mentioned earlier, existing context-aware platforms have proposed the use of conditions, rules or *if-then-else* statements to precisely specify the adaptation process. In contrast, XCML relies on a general context matching process [18] that selects the best matching version of each context-aware component with respect to the current client context. The decision to implement a best match rather than an exact match approach corresponds to trading off expressiveness for manageability. Rules and conditions give a developer full and fine-grained control over the context-aware adaptation process, but they require maintenance when new adaptation scenarios or new context dimensions need to be supported. Using XCML's context matching, developers have little control over the adaptation process itself,[5] but they can influence and configure it through the use of context expressions. In return for this loss of control, context-aware applications are more manageable as the introduction of both new component versions and new context dimensions become isolated operations that do not affect other existing parts of the application. Our experience with the development of complex context-aware applications, such as a content management system [2], a tourist information system [32] and the case studies discussed in this paper, has shown that this trade-off is reasonable as its expressiveness is sufficient. For example, the implementation of the tourist information system uses up to 25 context dimension with an average of 10 possible values each. This relatively large context space is required to support several client devices (home computer, festival kiosk, smartphone, paper-based client), different formats (HTML, VoiceXML, PDF), languages, interaction patterns [19], etc. In our experience, defining *if-then-else* statements to capture all possible context states implied by the context space is not feasible, nor is it maintainable if the context space changes during development.

Before formally defining the above three core concepts in the following subsections, we introduce a running example to both motivate and illustrate our approach. Figure 1 sketches a simple web site consisting of a single page. The content, navigation and presentation of our example web site are affected by four dimensions: language preferences, device characteristics, input modalities, and browser support. These four dimensions give rise to the *application-specific context model* and *context expressions* shown in Table 1. The language dimension determines whether English or German content is preferred. The web site is primarily designed for desktop computers, but provides a special version for small-screen devices such as mobile phones. The desktop version shows a banner on top and lets the navigation unfold

[5]The default algorithm of the general context matching process can be replaced with an application-specific implementation. However, the same algorithm is used for *all* context-aware adaptations within one application [18].

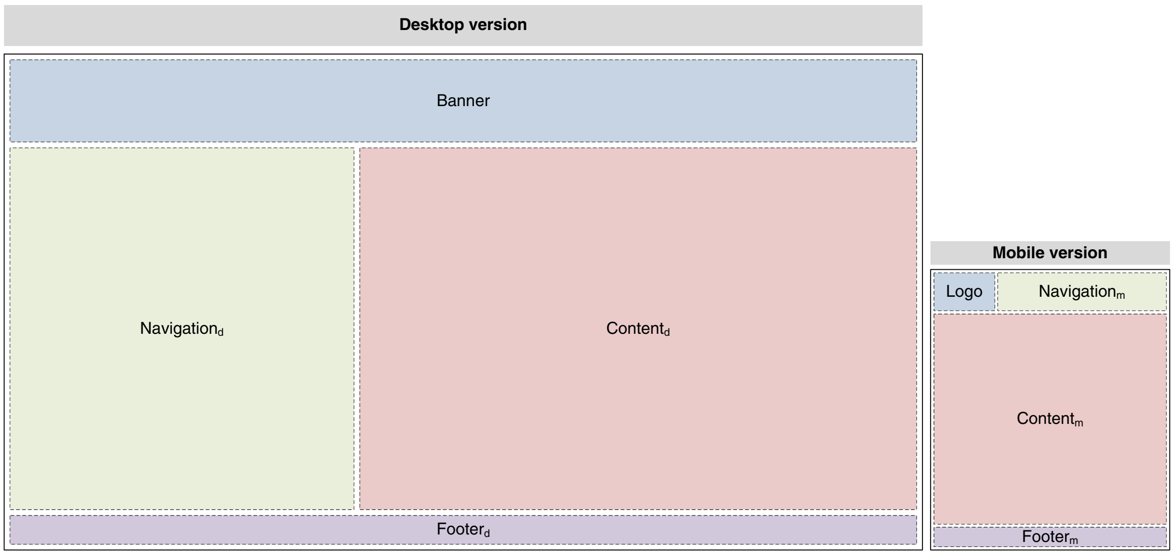


Figure 1 Example of a modern web site with a special version for mobile phones.

in the sidebar. To save some of the very limited screen space available on hand-held devices, the banner is substituted by a smaller logo in the mobile version and the navigation reduced to only show top level elements in a tabbed bar. The mobile version therefore adapts the web site to provide less content, shorter navigation paths and a simpler layout with minimum space requirements. Next to traditional input methods using mouse and keyboard, also touch is supported. The web site is developed for current versions of Internet Explorer and Firefox as well as providing legacy support for Internet Explorer 6. Note that the differences between the components of the desktop and mobile versions depicted in Figure 1 are indicated by indices d and m.

The *context-aware component model* is used to define content, navigation and presentation components. In our running example, the component representing the web page is composed of the four child components illustrated in Figure 1, i.e. header, navigation, content and footer. The header and content component each have two variants along the language dimension, one variant for English and one for German. The navigation component defines one variant for desktop and one for mobile platforms to adapt along the device dimension. Finally, the page

Table 1 Context dimensions and states distinguished in our web site example.

Context dimension	Context state	Context expression
Browser	{(*agent*, *"MSIE"*), (*version*, *"6"*)}	old_IE
	{(*agent*, *"MSIE"*), (*version*, *"7..9"*)}	new_IE
	{(*agent*, *"Firefox"*), (*version*, *"2..4"*)}	Firefox
Language	{(*lang*, *"en"*)}	English
	{(*lang*, *"de"*)}	German
Device	{(*screen-width*, *"640..1920"*)}	Desktop
	{(*screen-width*, *"1..480"*)}	Mobile
Input	{(*input-source*, *"mouse:keyboard"*)}	Mouse-Keyboard
	{(*input-source*, *"touch"*)}	Touch

component is associated with a presentation component which defines variants, again for the desktop and mobile contexts, to cater for the design differences indicated in Figure 1.

It is the task of the web application to provide the context model with the required context data. The language preferences used in the example above can, for example, be stored and retrieved using cookies or a registered user account linked to the web site. The browser and device contexts can be determined programmatically using the navigator and screen objects in JavaScript. Whether or not a device is touch-enabled can in some cases be derived from CSS media queries, e.g. -moz-touch-enabled in Firefox, or generally learned if touch events are fired instead of, or in addition to, mouse events. We discuss the components required to process and execute such an XCML application in Section 4.

3.1 Context model

The way in which context information is gathered, managed and augmented can vary vastly between different application domains. In order to be applicable in most domains, our approach needs to be as general as possible and therefore does not impose a particular context model. Instead, the only assumption we make about context is that it can be represented as a set of name-value pairs. We will show how this is general enough to handle quite sophisticated context models and is the basis for many context models used in practice which either use name-value pairs directly or can easily be mapped to this representation [3]. This general context representation is based on two sets, NAMES and VALUES, to denote legal context property names and values. First, we declare a context model that defines the property names relevant for an application.

Definition 1 The *context space* S of an application is a set of context property names, represented by

$$S = \{n_1, n_2, \ldots, n_n\},$$

where $S \subseteq \text{NAMES}$.

Based on the definition of S, we introduce the context representation based on name-value pairs that we motivated above. In our approach, this representation of context specifies the interface between the context-aware web application and a client used to access the site. This means that the client context must also be expressed in terms of the two sets, NAMES and VALUES.

Definition 2 A *context* of context space S is given by

$$C(S) = \{(n_1, v_1), (n_2, v_2), \ldots, (n_m, v_m)\},$$

where $n_i \in S \wedge v_i \in \text{VALUES}$ for $1 \leq i \leq m$ and $\forall\, (n_i, v_i), (n_j, v_j) \in C(S) : n_i = n_j \Rightarrow v_i = v_j$.

From the definitions above, it follows that a particular context specifies a set of context properties with concrete values, but not necessarily for all context property names in the context space. Note that this is possible as adaptation in our approach is based on selecting the version of a context-aware component that matches best, rather than exactly.

Having introduced this notion of context, we note that the general context representation used at run-time is not very practical for the specification of context-aware behaviour by developers. At design-time, XCML therefore introduces two additional concepts, namely context dimensions and context states. These concepts have been carefully designed to increase the expressive power of the context model at design-time, while keeping the simple run-time representation intact. Note that a key factor that enables us to have these two views of context is the fact that both representations are built from the same sets, NAMES and VALUES, and therefore no mapping between them is required.

Context dimensions are motivated by the fact that often only a subset of the context properties defined by S are relevant for a given component of a web site. As an example, imagine a context-aware text component that depends on the *language* property of the context, but not on, say, the browser *agent* or *version*.

Definition 3 A *context dimension* represents a set or semantic group of context property names that drive the adaptations of certain content, navigation and presentation components of a context-aware application. A context dimension D is defined as a set of names

$$D = \{n_1, n_2, \ldots, n_k\}$$

to distinguish k property names such that $D \subseteq \mathsf{NAMES}$.

In contrast to many context representations used in other approaches, e.g. [18, 35], we intentionally define a context dimension as a *set* of context property names, as this allows developers to group facets of context information and represent them as fine-grained as necessary to meet specific requirements of a context-aware application. For example, we could define a dimension $D_{\text{Browser}} = \{agent, version\}$ to distinguish, not only different user agents, but also different versions of the same browser, which is often necessary to work around cross-browser compatibility issues and to provide legacy support. Likewise, it can be useful to group together user-related aspects of the context model, e.g. current location and language preferences, as well as device characteristics, e.g. screen size and input modalities supported, in other dimensions.

In order to guarantee a consistent view of context at both design and run-time, we define how the context dimensions used at design-time relate to the context space S of the context-aware application.

Definition 4 The *context dimension space* of an application is represented by

$$\mathscr{D}(S) = \{D_1, D_2, \ldots, D_l\}.$$

The context dimensions in $\mathscr{D}(S)$ partition the context space S, i.e. $S = D_1 \cup D_2 \cup \cdots \cup D_l$ and $\forall\, D_i, D_j \in \mathscr{D}(S) : i \neq j \Rightarrow D_i \cap D_j = \emptyset$.

Also at design-time, XCML uses the concept of context states to match a range of run-time contexts that trigger the context-aware adaptations defined by the application. For each dimension, a particular context state associates values to all of the characteristics of that dimension. In line with previous work [18], the properties of a context state are represented as a set of name-value pairs, where the value of a context property is again a *set* of values. Note that single values can still be supported by using a set with only one element. As we will see later, this extended notion of context states is important in the development of context-aware applications because it allows components to be defined with adaptations that apply in several different run-time contexts. However, since it is often impractical or, particularly in larger context spaces, even impossible to explicitly specify the range of all context property values that trigger such adaptations, we further introduce a set notation that is used both as a shorthand in the following definitions and for expressions that are processed by XCML with the same result. For example, a simple enumeration such as "de:en" represents the set {*"de"*, *"en"*} for languages German and English, while an interval such as "1..3" is a placeholder for {1, 2, 3}. Also, enumerations and intervals can be combined to form expressions such as "1..3:5..8" to denote the numbers from 1 to 8 without 4. Finally, the wildcard * can be used as a shorthand to specify the set of all possible values. To accommodate these sets of values, we introduce $\mathsf{P\text{-}VALUES} = \mathscr{P}(\mathsf{VALUES})\backslash\emptyset$, i.e. the powerset of VALUES except the empty set.

Definition 5 A context state *CS* of a context dimension *D* is represented as a set of name-value pairs

$$CS(D) = \{(n, V) \mid n \in D \ \wedge \ V \in \mathsf{P\text{-}VALUES}\},$$

where $\forall\, n \in D : \exists\, (n, V) \in CS(D)$.

To illustrate the use of context states, let us revisit the running example introduced at the beginning of this section. In particular, we focus on the example of the context dimension D_{Browser} given above to explain how the context states given in Table 1 support the definition of a context-aware presentation component that depends on the browser used to access the web site. To refer to modern releases of Internet Explorer, i.e. versions 7 to 9, the developer could define the context state new_IE as {(*agent*, *"MSIE"*), (*version*, *"7..9"*)}. Similarly, a context state old_IE could be defined as {(*agent*, *"MSIE"*), (*version*, *6*)} to separate out prior versions for legacy support. Note that the value "MSIE" thereby denotes the user agent string returned for Internet Explorer. If the version of the browser is not important but the user agent is, the wildcard * can be used so that {(*agent*, *"MSIE"*), (*version*, *)} indicates that the context state will match all versions of Internet Explorer.

Listing 1 shows an XCML excerpt specifying the browser dimension and its corresponding states as given in Table 1. The code shows the definition of the Browser dimension in terms of agent and version. With the states defined thereafter, an application may distinguish between different versions of Internet Explorer and Firefox, i.e. old_IE or new_IE and Firefox. Note that very early versions of the browsers are not supported by this example web site. The above context model could easily be extended for new versions of these and other browsers by adding new context states to the Browser dimension. In the same way, the dimensions from Table 1 that were omitted from the above excerpt can be defined. Also note that separate definitions for dimensions make it possible to check whether states specify only context properties relevant for the corresponding dimensions.

```
<xcml:context-model>
 <xcml:context-dimension name="Browser">
  <xcml:context-key name="agent"/>
  <xcml:context-key name="version"/>
 </xcml:context-dimension>
 <xcml:context-state name="old_IE" dimension="Browser">
  <xcml:context-property key="agent" value="IE"/>
  <xcml:context-property key="version" value="6"/>
 </xcml:context-state>
 <xcml:context-state name="new_IE" dimension="Browser">
  <xcml:context-property key="agent" value="IE"/>
  <xcml:context-property key="version" value="7..9"/>
 </xcml:context-state>
 <xcml:context-state name="Firefox" dimension="Browser">
  <xcml:context-property key="agent" value="Firefox"/>
  <xcml:context-property key="version" value="2..4"/>
 </xcml:context-state>
</xcml:context-model>
```

Listing 1 Definition of a simple context model in XCML to distinguish different browsers.

3.2 Component model

While the context model establishes the context-awareness of an application, the component model specifies the context-adaptive features as well as the default behaviour of the application's components. Our approach is based on the fairly simple, but flexible model shown in Figure 2, where each component can define variants of content, structure and layout with respect to one or more context states.

The model separates the concepts of Component and Layout. Components hold arbitrary data and can be composed of other components. Each component is defined by a Type that defines its data in terms of attribute names and types. Examples of types range from simple content types such as text, link and image to application types such as customer, article and shopping cart. Finally, layouts provide presentation templates, namely XML documents, that specify how associated components are displayed in the client. Thus, components represent the content and structure of a web site, whereas layouts define their presentation. Both components and layouts

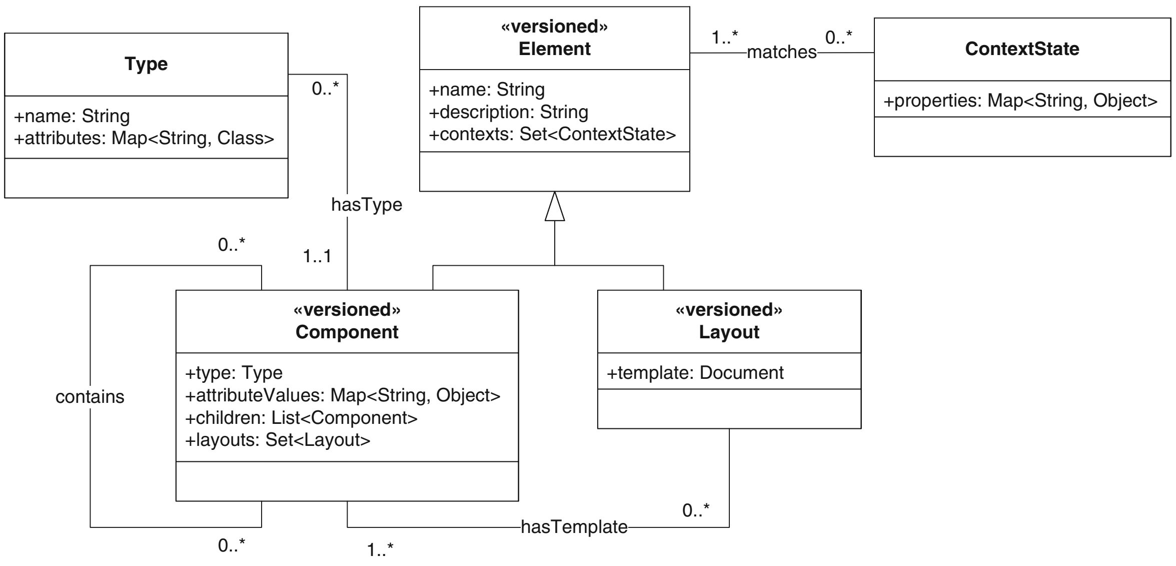


Figure 2 Component-based model used in XCML.

are specialisations of the concept Element that provides support for identifying and describing web site elements. Note that all three concepts—Element, Component and Layout—are defined using the stereotype ≪versioned≫. This stereotype is a short-hand to model objects that consist of one generic object and a set of concrete version objects [24]. Each version object is a valid representation of the generic object in a given context state, represented by the ContextState entity. Though this cannot be visualised graphically, associations can either reference the generic object or one of its versions. The hasType association is an example of the former case as it links the generic version of a Component to one Type. This captures the fact that all versions of a component share the same component type. The associations contains, hasTemplates, and matches are of the latter case and reference object versions. In the example of hasTemplates this allows each component version to be associated with a specific version of a layout template.

As a consequence, the classes of adaptation supported by XCML are adaptations of content, structure, and presentation. At the same time, adaptation can range from coarse to very fine grained, depending on how developers define context-aware components. For example, in order to support different page structures on different devices the component representing the page could be made context-aware. Conversely, to internationalise a single piece of text, it suffices to define additional versions of the corresponding text component. Generally, a component can have alternative contents and structures, and a layout can have alternative designs and output formats, with the best-matching version determined by the particular run-time context.

In the remainder of this subsection, we return to our running example and show how this separation of concerns applies in web applications based on native web technologies, such as HTML for content and structure and CSS for layout. The use of XCML in the setting of more advanced technologies is addressed in Sections 6.1 and 6.2, which respectively present how XCML can be integrated with existing web development platforms and extended to new adaptation scenarios. The former is shown based on OpenLaszlo which defines its own markup and uses different concepts to describe the user interface of a web application. The latter shows the use of adaptation to support large screen sizes based on HTML5 and CSS3.

Listings 2 and 3 show an excerpt from a possible XCML implementation of our running example based on HTML and CSS by including tags from the XHTML namespace. The structure of the web site in terms of content and navigation is defined in the code shown in Listing 2 and closely follows the description of Figure 1 at the beginning of this section. In particular, the code illustrates the definition of two context-dependent variants for both the header and the navigation component. The match clause specifies a context expression that will be used during evaluation to determine if the respective variant is a best-match for the current context. The link between the page component and its presentation template is defined using the layout attribute.

The presentation of the web page is specified in the excerpt shown in Listing 3 that defines the websiteLayout presentation template. To sketch the use of HTML and CSS in our approach, the template combines XCML with XHTML to place the web site's title in the best-matching language, link stylesheets optimised for Firefox or Internet Explorer depending on the browser context as well as adding visual enhancements for touch if the device is touch-operated. The HTML body then

```
<xcml:component name="website" layout="websiteLayout">
 <xcml:component-variant>
  <xcml:child-components>
   <xcml:component name="header" layout="headerLayout">
    <xcml:component-variant match="English">
     <xcml:attribute-value name="title" value="XCML Web Site Example"/> [..]
    </xcml:component-variant>
    <xcml:component-variant match="German">
     <xcml:attribute-value name="title" value="Beispielseite in XCML"/> [..]
    </xcml:component-variant>
   </xcml:component>
   <xcml:component name="navigation">
    <xcml:component-variant match="Desktop"> [..] </xcml:component-variant>
    <xcml:component-variant match="Mobile"> [..] </xcml:component-variant>
   </xcml:component>
   <xcml:component name="content"> [..] </xcml:component>
   <xcml:component name="footer"> [..] </xcml:component>
  </xcml:child-components>
 </xcml:component-variant>
</xcml:component>
```

Listing 2 Main component of example web site defined in XCML.

```
<xcml:layout name="websiteLayout">
 <xcml:layout-variant match="Desktop">
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML [..]//EN" [..]>
  <html>
   <head>
    <title> <xcml:attribute-value select="header/title"/> </title>
    <link href="default.css" rel="stylesheet" type="text/css"/>
    <xcml:context match="Firefox">
     <link href="firefox.css" rel="stylesheet" type="text/css"/>
    </xcml:context>
    <xcml:context match="old_IE or new_IE"> [..] </xcml:context>
    <xcml:context match="Touch">
     <link href="touch.css" rel="stylesheet" type="text/css"/>
    </xcml:context>
   </head>
   <body> <xcml:component select="*"/> </body>
  </html>
 </xcml:layout-variant>
 <xcml:layout-variant match="Mobile"> [..] </xcml:layout-variant>
</xcml:layout>
<xcml:layout name="headerLayout">
 <xcml:layout-variant match="Desktop">
  <img src="banner.jpg"/> [..]
 </xcml:layout-variant>
 <xcml:layout-variant match="Mobile">
  <img src="logo.gif"/> [..]
 </xcml:layout-variant>
</xcml:layout>
```

Listing 3 Layout of the example's main component using XCML in combination with HTML and CSS.

recursively includes all children of the associated website's child components, which will in turn be formatted by associated layouts. Due to space limitations, we only show the headerLayout as an example. With the variants defined for the header, the content becomes available in English and German and the presentation is adapted between the desktop and mobile versions to display either a banner or a smaller logo depending on the context.

3.3 Context expression language and algebra

The last two important components of our approach are the context expression language and algebra. In XCML, component and layout variants are associated with matching expressions along the context model to specify adaptive features with respect to the context states defined by the application. A simple example of such a context expression is new_IE to refer to the context state $\{(agent, \text{“}MSIE\text{”}), (version, \text{“}7..9\text{”})\}$ as defined earlier. While context states already provide the means to match a range of values defined in a particular dimension, XCML's context algebra introduces the following three important operations to formulate context matching expressions in terms of a combination of defined states, which may span more than one dimension.

Definition 6 Let $CS(D_1)$ and $CS(D_2)$ be two context states of dimensions $D_1, D_2 \in \mathscr{D}(S)$. The *conjunction* of these two states is defined as

$$\begin{aligned} CS(D_1) \wedge CS(D_2) = {} & \{(n, V_1) : (n, V_1) \in CS(D_1) \wedge \nexists \text{ some } V \text{ s.t. } (n, V) \in CS(D_2)\} \\ & \cup \{(n, V_2) : (n, V_2) \in CS(D_2) \wedge \nexists \text{ some } V \text{ s.t. } (n, V) \in CS(D_1)\} \\ & \cup \{(n, V) : (n, V_1) \in CS(D_1) \wedge (n, V_2) \in CS(D_2) \wedge V = V_1 \cap V_2\}. \end{aligned}$$

Definition 7 Let $CS(D_1)$ and $CS(D_2)$ be two context states of dimensions $D_1, D_2 \in \mathscr{D}(S)$. The *disjunction* of these two states is then defined as

$$CS(D_1) \vee CS(D_2) = \{(n, V) : (n, V_1) \in CS(D_1) \wedge (n, V_2) \in CS(D_2) : V = V_1 \cup V_2\}.$$

The first operation above can be used to build the *conjunction* over two states, i.e. the set of all context properties contained in either $CS(D_1)$ or $CS(D_2)$ is returned, but for the context property names that both states have in common, only the context property values contained in both states are included. The second operation defines the *disjunction* over two states so that, for the context property names that both states have in common, the set of all context property values contained in $CS(D_1)$ or $CS(D_2)$ are returned. These operations therefore allow for exclusive and inclusive combinations of context states. This is important since content, navigation and presentation of a web application are typically affected by more than one context dimension at a time. For example, browsers may provide either more or less features if they are deployed on a mobile device. A context expression, (Mobile and new_IE), can then be used to provide adaptations for the special case of using a modern version of Internet Explorer to access the web application from a mobile phone. This expression is a conjunction of two states from different context dimensions, for example, represented by $\{(screen\text{-}width, \text{“}1..480\text{”})\}$ for Mobile and $\{(agent, \text{“}MSIE\text{”}), (version, \text{“}7..9\text{”})\}$ for new_IE as before. In terms of the above operations, the expression would first evaluate to $\{(screen\text{-}width, \text{“}1..480\text{”})\} \cup \{(agent, \text{“}MSIE\text{”}), (version, \text{“}7..9\text{”})\} \cup \emptyset$ and subsequently to $\{(screen\text{-}width, \text{“}1..480\text{”}), (agent, \text{“}MSIE\text{”}), (version, \text{“}7..9\text{”})\}$. Note that the third element of the union in the first expression is the empty set because D_{Browser} and D_{Device} do not overlap. On the other hand, (old_IE and new_IE), with old_IE defined as $\{(agent, \text{“}MSIE\text{”}), (version, \text{“}6\text{”})\}$, would result in $\emptyset \cup \emptyset \cup \emptyset = \emptyset$, as these context states define different values in the same dimension and therefore all yield an empty intersection. Here, (old_IE or new_IE) must be used instead to yield $\{(agent, \{\text{“}MSIE\text{”}\} \cup \{\text{“}MSIE\text{”}\}), (version, \{6\} \cup \{7, 8, 9\})\}$ which, in turn, is equivalent to $\{(agent, \text{“}MSIE\text{”}), (version, \text{“}6..9\text{”})\}$. This means it only makes sense to use *or* between context states of the same dimension, while *and* is only meaningful in the case of context states from different dimensions.

While XCML also allows for nested expressions such as ((Desktop or Mobile) and (old_IE or new_IE)) using the same principles, particularly in larger context spaces, it can be useful to denote a match clause by negating a few context states rather than specifying all target states directly. The following operation is therefore defined to allow for expressions such as not Firefox that can act as a substitute for all the respective other states defined in the context space.

Definition 8 Let $\mathscr{D}(S) = \{D_1, \ldots, D_n\}$ be the context dimension space of an application. Further, $\forall i : 1 \leq i \leq n$ let $\mathscr{CS}(D_i) = \{CS^1(D_i), \ldots, CS^j(D_i)\}$ be the set of all context states defined in dimension D_i. The image $I(S)$ of context space S defined as

$$I(S) = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{|\mathscr{CS}(D_i)|} CS^j(D_i)$$

represents the set of context states covering the whole range of values for every dimension in S. The *negation* of a context state $CS(D)$ is then defined as

$$\neg CS(D) = \{(n, V) : (n, V_I) \in I(S) \wedge (n, V') \in CS(D) \wedge V = V_I \backslash V'\}.$$

The above operation first builds, for every dimension, the disjunction over all context states to yield a combined state covering the whole range of values in that dimension, and then the conjunction over the combined states for all dimensions. From the resulting image of the context space, it returns all states without the values of the given state. Note that the set of all context dimensions and, for each context dimension, the set of all defined states is implicitly given by the context model. To give an example, with the states Desktop defined as $\{(screen\text{-}width, \text{“}640..1920\text{”})\}$ and Firefox as $\{(agent, \text{“}Firefox\text{”}), (version, \text{“}2..4\text{”})\}$, as well as Mobile, old_IE and new_IE defined above, the expression not Firefox would translate to

$$\begin{aligned}
&\neg\{(agent, \text{“}Firefox\text{”}), (version, \text{“}2..4\text{”})\} \\
&\quad = \{(screen\text{-}width, \text{“}640..1920{:}1..480\text{”}), (agent, \text{“}MSIE{:}Firefox\text{”}), \\
&\qquad (version, \text{“}6..9{:}2..4\text{”})\} \backslash \{(agent, \text{“}Firefox\text{”}), (version, \text{“}2..4\text{”})\} \\
&\quad = \{(screen\text{-}width, \{640, .., 1920, 1, .., 480\} \backslash \emptyset), (agent, \{\text{“}MSIE\text{”}, \text{“}Firefox\text{”}\} \\
&\qquad \backslash \{\text{“}Firefox\text{”}\}), (version, \{6, 7, 8, 9, 2, 3, 4\} \backslash \{2, 3, 4\})\} \\
&\quad = \{(screen\text{-}width, \text{“}640..1920{:}1..480\text{”}), (agent, \text{“}MSIE\text{”}), (version, \text{“}6..9\text{”})\},
\end{aligned}$$

which, in this example, is the same as (Desktop or Mobile) and (old_IE or new_IE).

3.4 Context matching process

At the beginning of this section, we motivated why XCML favours application manageability over full control of the adaptation process. This trade-off is implemented using a general matching process [18] that compares the current run-time context to the context expressions defined for each version of a generic object. Based on this comparison, the matching process assigns a score to each variant. Finally, it automatically selects the highest-scoring variant as the context-dependent representation of the generic object. In order to exert control over XCML's context-aware adaptation process, a developer has the following configuration options, ranging from local to global.

- Required or illegal matches can be expressed by prefixing context values with a plus (+) or a minus (−) sign, respectively. For example, a version of layout template that requires Internet Explorer can be annotated using +(*agent*, *"MSIE"*). If the current run-time context does not contain this context value, that version of the template will not be selected, regardless of how high it scores otherwise. Conversely, a version of a text component that cannot be understood by German speakers can be annotated by −(*lang*, *"de"*) to prevent it from ever being selected when the run-time context contains that value. The elimination of versions that violate required or illegal matches from the matching process is achieved by setting their total score value to zero.
- Developers can assign weights to context dimensions that influence how versions of context-aware components are scored. If, for example, the *Device* dimension is more important to an application than the *Language* dimension, they could be weighted with 1.25 and 0.9, respectively.
- Weights can be defined for match classes. As discussed above, XCML supports the definition of context expressions that contain single values, sets or ranges. The general matching process can be configured to give a higher (or lower) score to component versions that match one specific value, than to those that match a set or a range of values.
- As a last resort, the algorithm that defines the general matching process is itself configurable. Therefore, it can be replaced with a custom implementation that is tailored to the requirements of a particular application.

As mentioned initially, we have built several complex systems based on our general context matching approach. For example, the EdFest tourist information system [32] supports paper, voice, and browser-based interaction, location-aware services as well as mobile and desktop clients. While all of the applications realised so far make use of the first three configuration options and, in particular, rely on the possibility to express required and illegal matches, none of these systems required the general matching algorithm itself to be replaced with a customized. Even though these experiences can only serve as empirical evidence, they nevertheless indicate that the expressiveness of our approach is sufficient to build advanced systems.

3.5 Representation in XML

In previous subsections, we have used code excerpts to demonstrate the use of XCML concepts based on our running example. To conclude this section, we now provide a full description of the language in terms of its metamodel and syntax. Our implementation of XCML combines the concepts above and integrates them tightly with XML. We have defined both an XML schema and DTD for the XCML language components on top of XML. While the schema gives full control over XML namespaces, the DTD is often sufficient for applications where code validation has the highest priority. To show how the proposed concepts translate to XML, Listing 4 contains the main components of the DTD that we defined. Note that the concrete attribute list definitions have been omitted in order to focus on the structure of XCML documents.

```
<!ELEMENT xcml (context-model?,(type|layout|component|import)*)>
<!-- content model -->
<!ELEMENT context-model (context-dimension|context-state)*>
<!ELEMENT context-dimension (context-key+)>
<!ELEMENT context-key EMPTY>
<!ELEMENT context-state (context-property+)>
<!ELEMENT context-property EMPTY>
<!-- types -->
<!ELEMENT type (attribute*)>
<!ELEMENT attribute EMPTY>
<!-- components -->
<!ELEMENT component (component-variant+)>
<!ELEMENT component-variant (attribute-values?,child-components?)>
<!ELEMENT child-components (component+)>
<!ELEMENT attribute-values (attribute-value+)>
<!ELEMENT attribute-value (#PCDATA)>
<!-- layouts -->
<!ELEMENT layout (layout-variant+)>
<!ELEMENT layout-variant (#PCDATA|match-context|select-component|select-
    attribute-value)*>
<!ELEMENT match-context ANY>
<!ELEMENT select-component EMPTY>
<!ELEMENT select-attribute-value EMPTY>
<!-- imports -->
<!ELEMENT import EMPTY>
```

Listing 4 Document Type Definition of XCML documents without attribute list definitions.

As can be seen from the DTD, XCML documents mainly consist of four components: the context model, followed by type, component and layout definitions. The context model is defined in terms of dimensions and states where dimensions define the context keys and states allocations of these in the form of a property list or name-value pairs. Type definitions are rather simple and essentially list attributes together with their base types, e.g. text, boolean, integer (not shown here). Components can then be defined based on such type definitions in different variants which in turn can contain other components depending on the context. For each of these components, layouts can be defined again in different variants, using the respective select elements for components and attribute values to be computed for the current context and added to the output. Also important is the match-context element that allows parts of a layout template to be defined only for certain contexts. We have shown examples of this in Listing 3. Finally, import can be used to include other XCML documents that can extend the context model and provide additional type, component and layout definitions.

4 Execution environment

As one possible execution environment for XCML, the implementation presented in this section is based on a generalisation of the context-aware data management system developed in previous work [18]. The decision to build on this infrastructure for the XCML compilation and execution process, i.e., not just to store the data managed by a web application, but to manage and run the application itself, is not just the result of an intention to minimise the development effort for the proof-of-concept, but also brings several general advantages. First, the database system can improve the performance, especially for larger context spaces, as it better facilitates storage and retrieval of complex components and layouts with many variants. Second, given an initial context model and primary context-aware web page elements, an application can evolve dynamically without the need to regenerate the entire site. For example, new components or variants can be added to the database at run-time and existing ones updated or deleted as required. To this end, we have defined several components for the processing of XCML, which now form the new *design-time* and complement the *run-time* components also used in previous work.

The resulting platform supports two processes: (1) the compilation process of XCML to create context-dependent variants in the database and (2) the linking process triggered in response to client requests and the context in which these take place. The roles of the individual system components are as follows.

Database builder As the Database Builder parses the XCML documents, it employs the Context Builder to build the context space from the model and creates the corresponding context-dependent variants in the database. We describe this rather complex process in more detail later in this section.

Context engine The Context Engine has been extended with the Context Builder as part of the design-time to provide several methods to parse context information declared in XCML documents. The Context Algebra is represented by a utility class providing implementations of the formally defined operations in Section 3. The role of the Context Matcher is to score all variants of a specific object and to determine the best-matching version for a given context at run-time.

Page builder In response to client requests, the Page Builder uses the Context Matcher to retrieve the best-matching variant for all context-aware elements stored in the database and assembles them into the requested version of the web site.

4.1 Design-time process

We first focus on process (1) in Figure 3 and explain how XCML documents are processed and compiled into the database. As illustrated in Figure 4, the processing of presentation instructions within xcml:layout tags is handled differently from processing XCML component variants.

Processing of components For each xcml:component-variant, the Component Builder creates a version with the respective attribute value sets. Each version is appended to its component object and stored in the database together with the specific context that was evaluated from the associated match clause. This step repeats recursively for all child components contained in each variant.

The processing of components is rather straightforward as we specify each variant separately using structured data in XCML attribute values. We have, however, defined an inheritance mechanism for component variants to inherit from the default variant using the copy-default attribute and only override the context-sensitive values. This kind of inheritance is reasonable as default variants are usually rich in detail, and some fields, such as a person's first and last name in staff member pages, are constant across all versions. This is in contrast to layouts where the underlying templates are typically semi-structured and patterns for re-use are somewhat

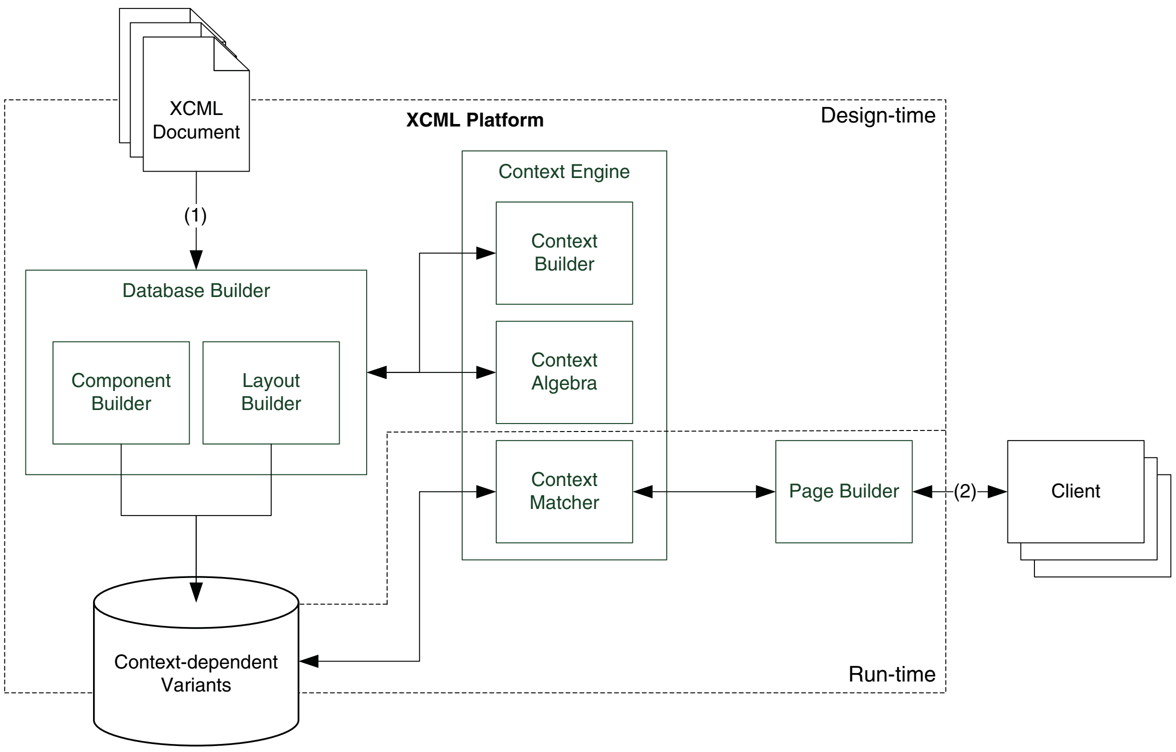


Figure 3 Integration of the new design-time components with XCM run-time.

different. For that reason, we allow the developer to define layout variants separately similar to components but without inheritance, or to use nested context matching expressions within the same layout variant. In addition, developers can use the import statement to include templates defined in another XCML document. If the

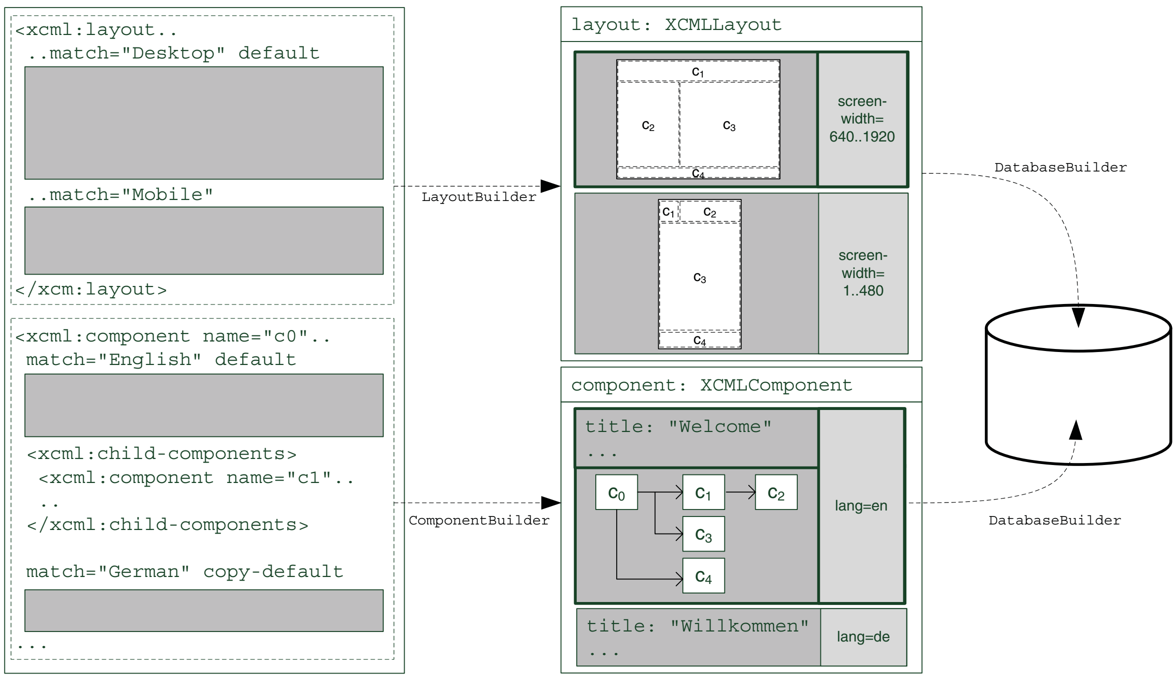


Figure 4 Processing of XCML layouts and components.

differences between individual templates are not substantial, then nested variants are more practical in order not to duplicate large parts between versions that are essentially the same. The processing of XCML templates as implemented by the Layout Builder may therefore require further steps to compile any nested variants.

Processing of layouts For each xcml:layout defined in an XCML document, the Layout Builder pre-processes all specified variants to find, combine, and evaluate the individual context expressions that may have been nested inside the template. For each context evaluated from the expressions, it then runs a first XSL transformation on the template to generate a new XSL template that

(a) copies only the XML child nodes nested in matching xcml:context blocks and
(b) replaces each xcml:attribute-value/xcml:component select with a corresponding xsl:value-of select statement.

Step (a) therefore reduces a multi-dimensional XCML template to an XSL template specific to the context for which it was evaluated. Step (b) is required to resolve the XCML namespace and prepare the resulting XSL template for the run-time where all placeholders for nested attribute values and components will be substituted with the best-matching results evaluated for the specific context. Finally, each of the reduced XSL templates will be appended to the corresponding layout object and stored in the database together with the context.

4.2 Run-time process

At run-time, the Page Builder and the Context Matcher interact with the database to render context-dependent pages. This process, labelled (2) in Figure 3, has been discussed in detail in [18]. In order to be self-contained, we briefly summarise the key points of the matching process. Figure 5 sketches the content of the context-aware database after process (1) has completed for the running example used in Section 3. The data model used in the figure to visualize the database content corresponds to the component model shown in Figure 2. For the sake of presentation, we have chosen this very simple example as it only consists of one multi-lingual page that can be accessed on desktop and mobile devices. Nevertheless, it is sufficient to illustrate XCML's run-time process.

The page is stored as an instance of Component, named `website`, and linked to an instance of Layout, named `websiteLayout`, through the hasTemplate association. Since the application developer did not explicitly define variants of the page, it only has a *default* version. In contrast, the database contains two versions of the associated layout template that match context state Desktop and Mobile, respectively. In accordance to the page structure shown in Figure 1, the database also contains the four subcomponents of the page, namely `header`, `navigation`, `content` and `footer`. The subcomponents are linked to the default version of the page through the contains association and most of them have several variants themselves. For conciseness, we only show the layout template for the header component.

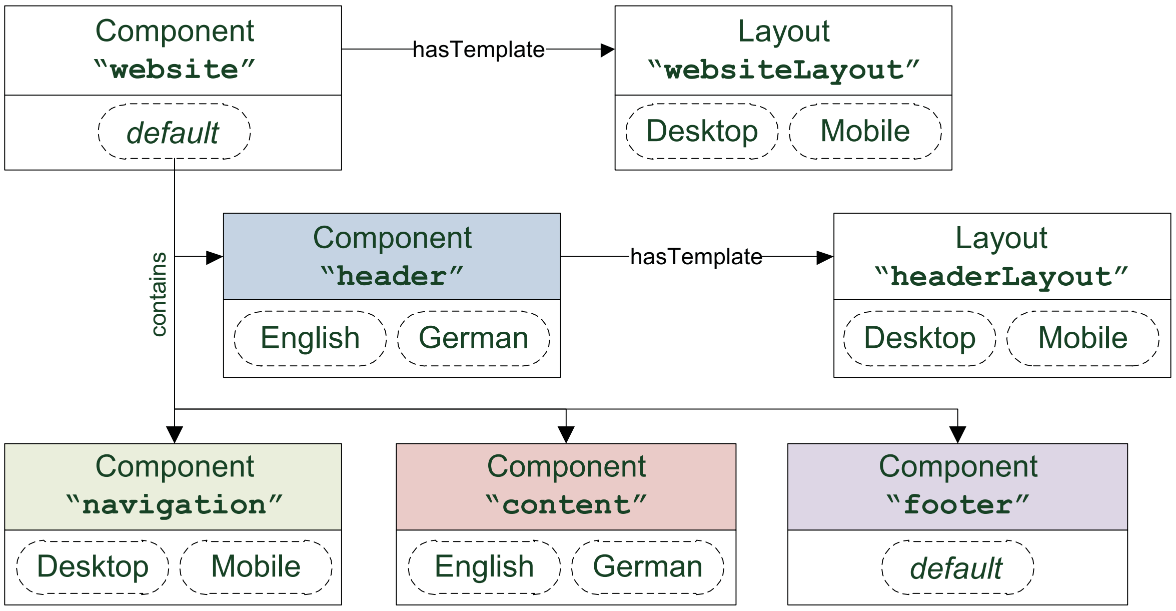


Figure 5 Database content after the compilation of the XCML example application.

To describe how the XCML run-time supports context-adaptation, let us assume a client with context

$$C(S) = \{(agent, \text{"}MSIE\text{"}), (lang, \text{"}en\text{"}), (screen\text{-}width, \text{"}1920\text{"}), \\ (input\text{-}source, \text{"}touch\text{"})\}$$

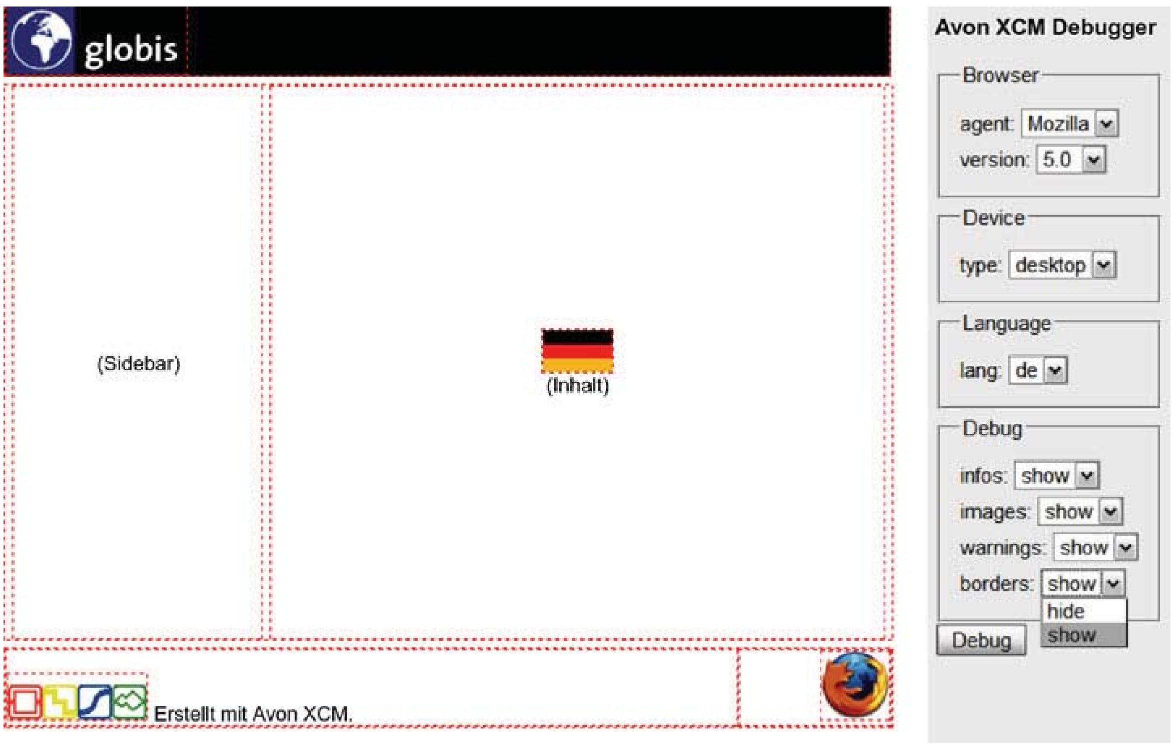
requests the example page. To respond to this request, the XCML run-time recursively processes the tree of components and builds an XML document. In parallel, it processes all the associated layout templates and includes them in an XSLT stylesheet. Once both documents have been assembled, they are used to generate the page that is sent back to the client. Whenever a component or a layout is considered for inclusion in either of these two documents, the XCML run-time uses the client context $C(S)$ to assign a score value to all of its versions. For example, when processing the header component, it will select the English variant as it is a 100% match, rather than the German variant that scores 0%. Clearly, our simple example has been designed to be "well-behaved" in order to illustrate the general matching process used in XCML. In Section 3.4, we have discussed possible solutions how developers can deal with situations where multiple versions obtain the maximum score or when no version scores sufficiently high. These situations can occur when a variant has been defined for several context states, but only matches each of these states partially.

5 Development support

To support developers in the design and specification of context-aware and adaptive applications based on XCML, our development environment leverages several visual tools that are also context-aware and aim at assisting developers in the authoring and testing of adaptive web applications. Many of these address basic requirements to support the language-based approach of XCML that would otherwise lack support for more visual representations inherent to many model-based approaches, e.g., for the context and adaptation model. For example, we have extended the authoring environment with design-time support for XCML applications by providing a context view for developers to define and set the target context during development based on the context model defined for an application. The authoring environment is sensitive to changes in the context view so that only the components of an XCML application that match the current design context will be active and available for editing. This makes it easier to manage more complex adaptive applications, as it helps developers to focus on selected parts of an application depending on their role in the development process and to build and execute only these parts without the need to compile the entire application. In the following, we want to focus on a context-aware debug component that can be embedded and customised as part of an XCML application at any stage of the development process. This is an important requirement and a tool worth mentioning since the testing and debugging of adaptive applications has received only little attention in the literature.

To support build and test cycles in a systematic way, our new idea was to build on the context-aware concepts from Section 3 and to allow developers to define a specific test mode based on a separate context dimension. The context states defined for this dimension can be used to activate and configure the debug tool as well as to control debug-related output and graphical annotations of corresponding web site components as they are rendered in the browser. The debug component can be integrated into a web site either at a global level for all versions, or directly in a particular variant for local testing only. Figure 6 illustrates the use of such a debug component at a global level.

The web site shown in this example can be adapted along three context dimensions, i.e. language, browser and device. For the debugging, it defines an additional



Browser
agent: Mozilla
version: 5.0

Device
type: desktop

Language
lang: de

Debug
infos: show
images: show
warnings: show
borders: show

Figure 6 Troubleshooting an example web site with our context-aware debugger.

dimension $D_{\text{Debug}} = \{\ infos, warnings, borders, images\ \}$. As a result, the debug component integrated with the web site provides the following features for systematic testing that are derived from the context model.

Context switching The context options provided by the debug component allow developers to select and refine the run-time context. This enables switching between different contexts and simulation of possible run-time situations. For example, the developer can quickly switch between the desktop and the mobile version of the web site and test them in different languages as well as for different browsers.

Debug output The available options also allow to control the debug output so that warning messages and debug information as generated by the execution platform are either included or excluded from the output. This can be helpful to test the application at different levels of error reporting and to smoothen the transition between debug and release builds.

Visual aids The debug output can further be complemented with debug-related style definitions that will be applied according to the options selected in the debug component. In particular, the option to show borders is helpful to identify the separate components that a web site is composed of and to locate errors with respect to the structure and presentation of a web site.

Custom debug states While the debug options mentioned so far are generally useful for many debugging scenarios, for this particular application, the option to hide images is included primarily for the mobile version. Mobile users sometimes choose to block images in order to achieve better load times and so it is important to test and verify that the design and layout of the web site also works in such a case.

6 Case studies

Having presented the concepts and mechanisms behind XCML, a first application and how it is processed by the execution environment as well as the tool support for developers, we will now focus on the evaluation which we present using two examples. The first shows a simple integration of XCML with existing platforms that often only provide a limited notion of context. We will use OpenLaszlo as an example and, in particular, demonstrate how two separate implementations of an existing OpenLaszlo application—one for the desktop and the other for mobile devices—can be integrated into a single, context-aware application with support for both platforms. The second case study addresses support for new adaptation scenarios in existing applications using a systematic approach based on XCML. We will focus on large-screen adaptation as a new challenge that has so far received little attention. Also we will address the required forms of adaptation first using native web technologies HTML5 and CSS3 only. While this is used to demonstrate that the new web standards provide support for some of the key adaptation techniques that previously required more comprehensive conceptual approaches, it also shows the limitations of this technological approach. The scenario is therefore used as an example of where XCML could be used to complement and extend the support of existing technologies at the language level. It is important to note that, while both examples address the same problem of multi-channel access using XCML, the initial situation is different. The reengineering of existing non-adaptive applications and the extension of a site with additional adaptation support are two separate issues that pose different challenges. At the same time, they go hand in hand and it is therefore interesting to look at both examples together and not only in isolation. The two cases are therefore followed by discussion where we assess the development effort and design simplicity for the developer and the new opportunities using XCML.

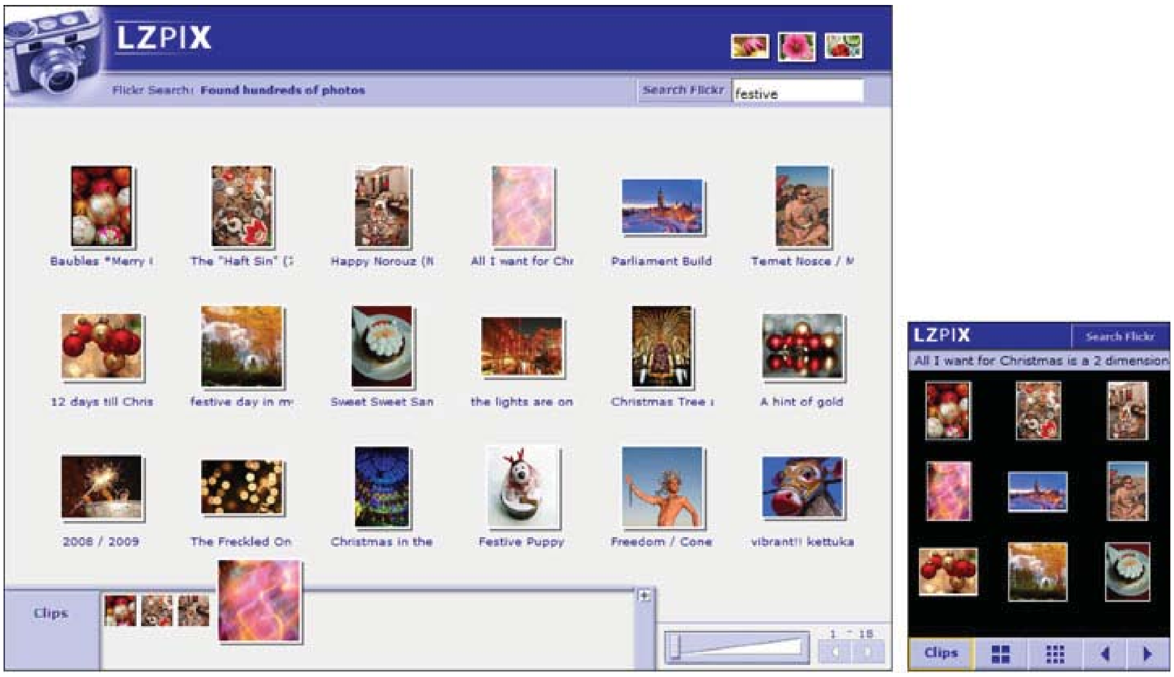
6.1 Integration with existing platforms and applications: LZPiX case study

The first case study was designed around a scenario that requires the integration of XCML with existing platforms and applications. It is important to allow for an integration with existing solutions because the adaptation to different devices and platforms, e.g. in the case of desktop-to-mobile adaptation, often involves switching between different technological setups. For the same reasons the integration is also challenging, as it has to be supported, not only at the language level for the design-time, but also at the implementation level of the platforms used at run-time.

To demonstrate the extensibility and portability of XCML and its execution environment, we will focus on the integration with OpenLaszlo for two reasons. First, as is the case for the majority of user interface description languages, the underlying language, Laszlo Script (LZX), is based on XML and this simplifies the integration with our current implementation of XCML. Second, OpenLaszlo is an open-source framework with an active developer community that maintains several showcase applications.[6] One such application is LZPiX, which is particularly interesting because it comes in two versions, LZPiX Desktop (Figure 7a) and LZPiX Mobile (Figure 7b), with the latter optimised for smaller screen real estates and directional navigation via a context-sensitive menu bar. The fact that these two versions are indeed separate implementations makes LZPiX a good example for which we can show that an integration of the proposed context-aware concepts is feasible and brings the important advantage of having to maintain only one codebase.

As illustrated in Figure 7, LZPiX is a web application that displays pictures retrieved from Flickr in a rich, desktop-like user interface. Users can search Flickr, browse through the photos returned over multiple pages and manage their favourite images in a separate clips view. Both versions also support changing the size of the thumbnails displayed, but the most obvious difference is that LZPiX Mobile requires many more separate screens to display the information that can fit on one screen in the desktop version. To get a better understanding of how the two versions differ in terms of the implementation to then see what is effectively required to develop a single, context-aware LZPiX application using XCML, our case study involved the following three steps: (a) structural and semantic analyses of LZPiX Desktop and LZPiX Mobile, (b) the unification of the two codebases to group shared logic and identify starting points for context-dependent variants and (c) the final reengineering of the separate LZPiX implementations into an integrated, context-adaptive web application.

[6]http://www.openlaszlo.org/showcase



(a) Desktop (b) Mobile

Figure 7 Screenshots of OpenLaszlo's LZPiX showcase application.

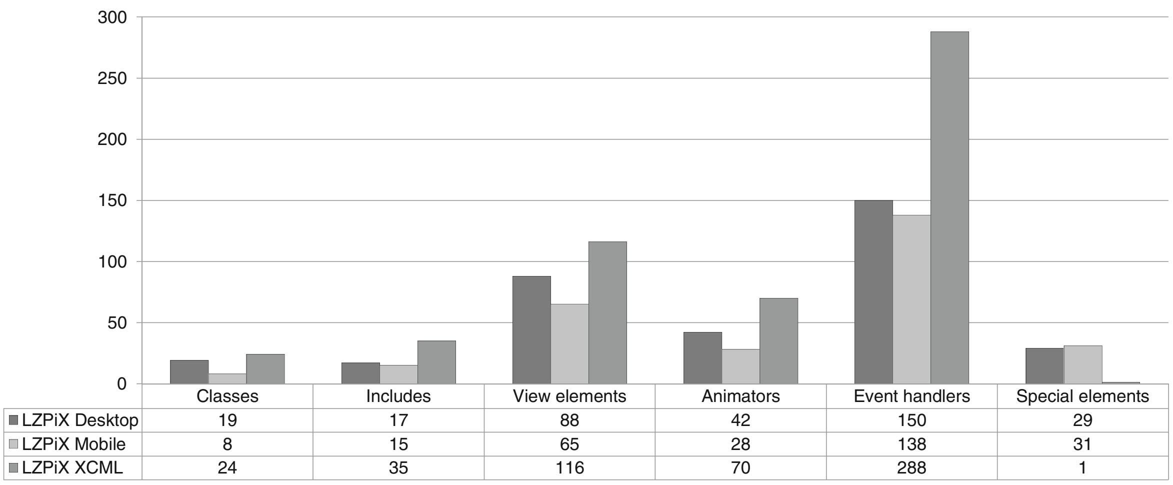
For the first step, we developed an LZX code parser to perform a structural analysis on the source files of LZPiX Desktop[7] and LZPiX Mobile.[8] Our investigations revealed that the underlying codebases are fairly different in terms of naming and packaging, and thus only share a few OpenLaszlo classes, i.e. photo, search, taglink, and includes, i.e. dataman, photo, resources. The results shown in Figure 8 indicate that LZPiX Desktop is a lot more complex than the mobile version, consisting of 32% more LZX elements (545 in LZPiX Desktop as opposed to 373) with generally more classes, includes, view elements, animators and event handlers. Note that "special elements" in Figure 8 refers to those view elements that are specific to only one of the two implementations.

The semantic analysis, on the other hand, showed that both implementations essentially consist of four common components: a search component to dispatch queries to Flickr, a thumbnail view of the retrieved photos, a details view showing a larger version of selected pictures and metadata associated with them, and finally a clips view to manage favourites. The main differences are that the desktop version provides an extra screen at startup with predefined Flickr queries, supports drag-n-drop between the thumbnail views and the clips, and generally makes heavy use of animations for smooth transitions between different views. In contrast, LZPiX Mobile splits some of the views over two or more separate screens and uses a context-sensitive menu bar for navigating between screens.

Relatively simple refactorings of the original LZPiX sources allowed us to form a common codebase with 24 classes (with versions of photo and search), 35 includes

[7]http://www.openlaszlo.org/lps_demos/demos/LZPiX/app.lzx

[8]http://www.openlaszlo.org/lps_demos/demos/LZPiXmobile/main.lzx



	Classes	Includes	View elements	Animators	Event handlers	Special elements
■ LZPiX Desktop	19	17	88	42	150	29
■ LZPiX Mobile	8	15	65	28	138	31
■ LZPiX XCML	24	35	116	70	288	1

Figure 8 Results from the structural analysis of the two LZPiX implementations and the resulting XCML application.

(with versions of dataman, photo and resources) and 116 view elements (Figure 8). This means that the reengineering process allowed us to reduce the overall complexity in terms of view elements for the two separate implementations by almost 25% since they can now be shared between versions and therefore there is no need to maintain them separately. For the XCML implementation, we first defined a one-dimensional context model for the Device with two states, Desktop and Mobile, to represent the desktop and mobile settings similar to Table 1. We then reorganised the main module of both versions along the four common components and used XCML to specify the context-dependent variants between them. Subsequently reengineering the separate LZPiX implementations required a total of 6 components, i.e. LZPiX, search, photos, details, clips, menubar, with 7 component variants and respectively 6 layouts with 11 variants. These relatively small numbers speak for the fact that, after our careful analyses and the unification of the codebases, only a few adaptations had to be made to integrate the two LZPiX implementations. The higher number of layout variants was required to preserve the aforementioned "special elements" of each version. Finally, we note that our adaptations mostly concerned the top level view elements. While a deeper structural analysis could identify more commonalities and differences also within these elements, lower level adaptations may only make sense if they do not result in too many small variants.

In the last step, we extended the execution environment presented in the previous section to achieve an integration with OpenLaszlo as shown in Figure 9. At design-time, XCML is used in combination with LZX (1) to produce the OpenLazlo markup shown in the examples above, which only required to take care of the namespaces since both languages are based on XML. The more difficult step was the integration of the XCML run-time components with OpenLaszlo's compilation process. Our current solution divides process (2) of the XCML platform (Figure 9) in two subprocesses. First, the LZX code is retrieved in the best-matching version with respect to the current context (2a) and then OpenLaszlo is called to render the context-dependent result (2b). While a tighter integration could provide a better performance in terms of the required processing steps, this kind of integration is relatively simple to achieve and works as a proof-of-concept.

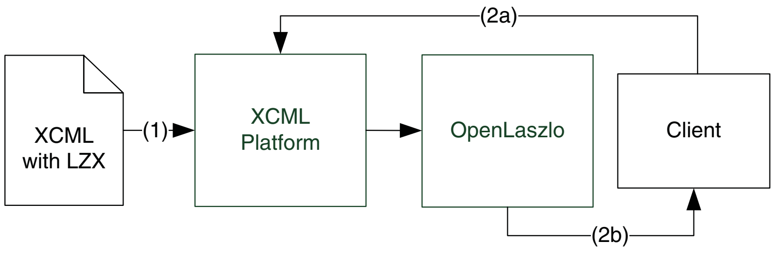


Figure 9 Integration of XCML and its execution environment with OpenLaszlo.

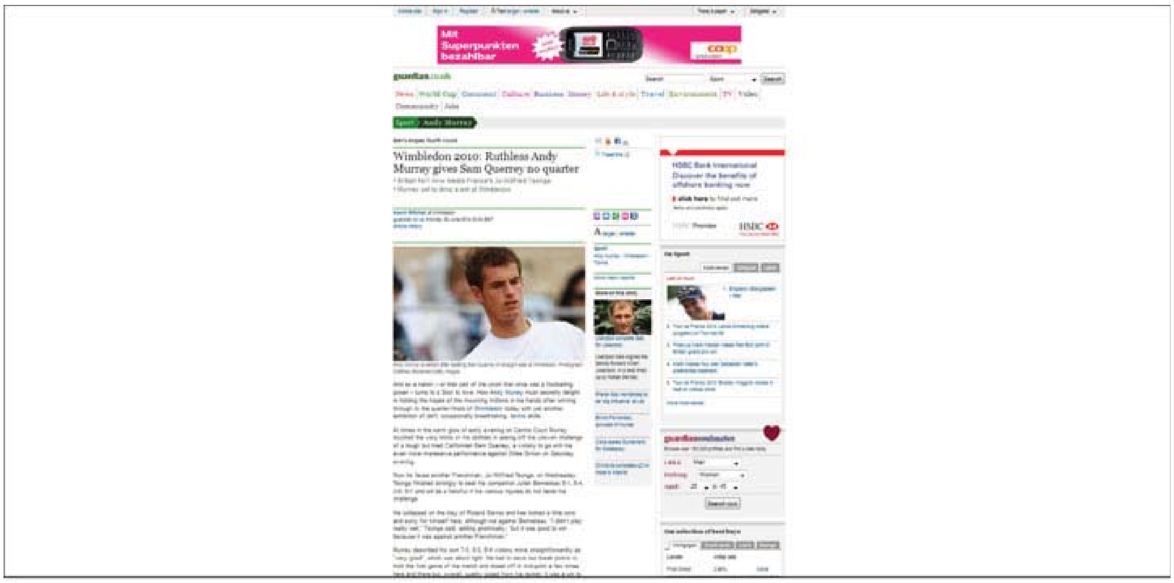
6.2 Extension for new adaptation scenarios: the guardian case study

The focus of our second case study was on systematic support for new adaptation scenarios and the extension of context-awareness in applications to address additional requirements as they emerge. Given the trend of increasingly large display sizes and resolutions, one such scenario is the adaptation to larger screen real estate which the majority of today's web sites fail to address. For example, most news web sites provide access to content in various forms often including a mobile version of the web site; however, a higher quality version for large-screen applications is typically not one of them [28]. The case study presented here will focus on the Guardian online newspaper[9] as an example of a design and structure common to many news sites. We will present a solution based on XCML and HTML5/CSS3 that adapts the content and presentation of this and similar information-centric web sites for widescreen contexts. The problem was approached in two steps. First, we used only web standards with the intention of measuring the level of support with native web technologies and to explore their limitations. In the second step, we complemented and extended this approach with features of XCML and compared both techniques in terms of development effort and design simplicity for the developer.

Figure 10a shows the Guardian's original design viewed on a 30″ screen at a resolution of 2560×1600 pixels. The content comprises the typical web page elements, such as the header containing the main navigation bar and a slot for advertisements at the top, followed by the main content and the footer at the very bottom of the page. The main part is presented in three columns: the leftmost column that contains the news article content, a smaller column in the middle which provides various user functions and related information, and the far right column that shows advertisements and links to related pages and various services. As is evident from the screenshot, the layout does not adapt well to the example viewing situation with the effect that a considerable amount of screen space left and right is unused. Also the text appears smaller on the large screen since it was intended for a lower DPI ratio. Finally, the media as well as the ads do not scale well with the remaining content.

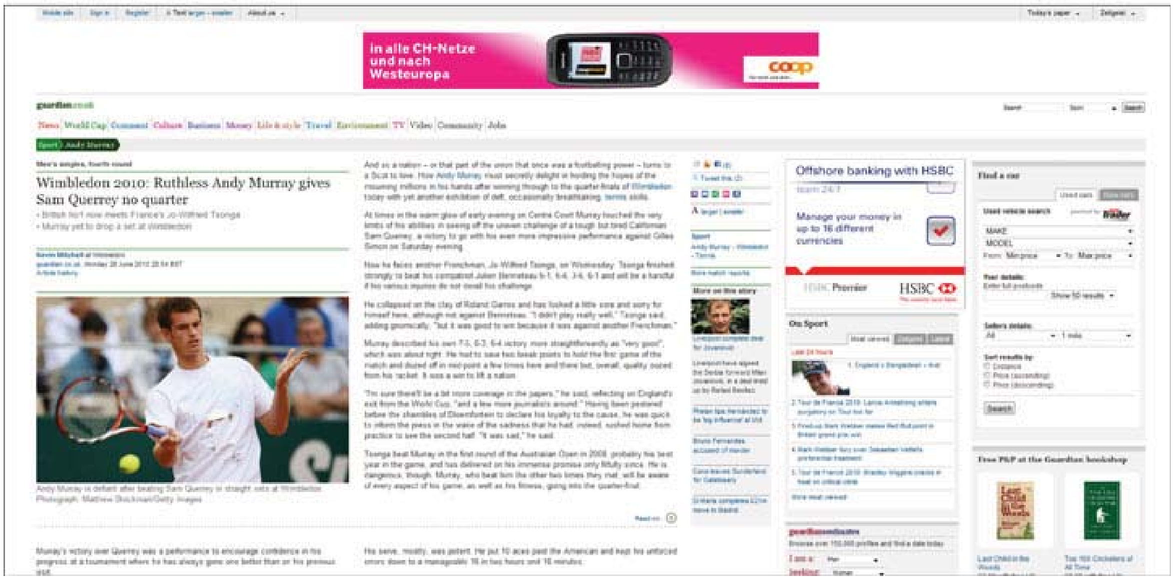
The goal of the first step was therefore to adapt the layout so that it is flexible and scales more effectively at larger viewing sizes. To achieve this, we have first carried out an extensive review of the new features of HTML5 and CSS3 and determined the kinds of adaptations that need to be supported to accommodate larger screen real estates. We have then implemented an adaptive layout template and applied it to the Guardian web site, the result of which is shown in Figure 10b. The adaptations include automatic scaling of font and media, the controlled use of multi-

[9]http://www.guardian.co.uk/



Wimbledon 2010: Ruthless Andy Murray gives Sam Querrey no quarter

(a) Original design of the Guardian online newspaper



in alle CH-Netze und nach Westeuropa

Wimbledon 2010: Ruthless Andy Murray gives Sam Querrey no quarter

Offshore banking with HSBC

Manage your money in up to 16 different currencies

(b) Adapted version for large, wide-format viewing situations

Figure 10 Examples showing the Guardian's news web site in (**a**) its original design and (**b**) with adaptations for widescreen contexts using new features of HTML5 and CSS3.

column layout and higher quality multimedia content—all of which are supported by client-side technologies HTML5 and CSS3 in combination with media queries and JavaScript. To give an example, we used media queries to specify different sets of styles for larger window or display sizes that will only be applied in widescreen contexts. The concrete design decisions and the adaptation operations necessary to accommodate large-screen environments are further detailed in [29].

The study and this first approach were interesting as they revealed several shortcomings of current web standards due to the limited notion of context. There were two key issues that posed a major problem in the case of the Guardian. First, current concepts and techniques such as media queries only enable context-dependent variation of the presentation in terms of layout and design, but are insufficient if required adaptations also need to concern the content of the web page. For example, this became evident when we wanted to embed multimedia content in different versions to be able to switch to higher quality content for larger viewing sizes. We have experimented with different approaches, e.g. to embed two multimedia objects in HTML and toggle their visibility using media queries, or to specify a placeholder in HTML and select the multimedia content via CSS. The problem with the first approach is that both objects will be transferred to the client and the second means that the specification of content and layout will be mixed in CSS. A third approach was again based on the placeholder in HTML and required JavaScript to load the content in the corresponding version. However, this alternative raised a second issue, namely that there is no uniform method for the specification of context across languages. In particular, JavaScript does not provide the methods for directly accessing device characteristics in the same manner as CSS media queries. We therefore had to use the workaround of dynamically creating hidden elements at run-time and associating them with different styles and conditions using media queries, to then check with JavaScript whether the changes have applied. Recently, some vendors have started to address these issues by providing a native, but again browser-specific, method for evaluating media queries and executing associated logic also in JavaScript.

In the second part of our study, we have therefore used XCML to alleviate some of these problems, which required two steps. First, we defined a simple context model to extend the original web site with the new version created for large-screen contexts using two states, wide and ultra-wide, for horizontal screen distances of 1,280 and 1,920 pixels or higher. In terms of the adaptations, the wide layout adds columns for the text and the ultra-wide one uses multi-column layout also for the sidebar controls as shown in Figure 10b. In the second step, we linked the large-screen layout of the web site to the main component for the above states and declared the original Guardian web site as the default version for other screen contexts. XCML made it fairly easy to declare the higher-quality variants of the multimedia content and to make sure that only matching content will be delivered to the client. We used the concept of nested layout variants to express the slight differences between the two widescreen versions, and this approach required no workarounds based on JavaScript that were necessary in the first approach. In particular, the combined use of media queries and XCML made for a highly-adaptive solution using a combination of client-side and server-side adaptation based on the same simple context model.

7 Discussion

The two case studies presented in the paper served several purposes. First, the feasibility of our approach and the usability of the language concepts were demonstrated in the scope of two existing real-world applications. Second, we have outlined the blueprint of one method for reengineering separate versions of a web site into a single, context-aware application and another one for using XCML to extend an existing web site with new adaptations. Finally, we have explored the support for context-awareness in state-of-the-art technologies such as HTML5 and CSS3 and identified current limitations and situations where XCML could be of particular benefit.

Looking at the examples in more detail, the first uses OpenLaszlo as an example to show how our approach can be generalised to existing platforms and combined with other languages. In particular, looking at the engineering effort required to integrate XCML's concepts for context-awareness in a non-adaptive application such as LZPiX that uses two separate implementations for desktop and mobile platforms is interesting since the scenario is representative for many existing web applications that come in different editions. The second case study has further underlined the flexibility of our approach since we show how an existing web site such as the Guardian can be systematically extended to support new adaptation scenarios. One of the main advantages of our solution based on XCML is that there is no need for modelling or reengineering the original site in order to adapt it. This is a major difference to many model-driven approaches, such as WebML or AMACONT, that would first require a specific model or description of the user interface before it can be retargeted. In terms of design simplicty for the XCML developer, both use cases only required the definition of a minimal context model and the controlled linking of different component and layout variants. In particular, the first example also reduced the complexity of the application in terms of view elements required to support both the desktop and mobile platforms, and the second scenario was fairly easy to support without adding much to the complexity of the existing application. While we can therefore see potential advantages for XCML to generally simplify the design process, the main potential lies in the support for the evolution and maintenance of XCML applications since they could now serve as the starting point for more adaptive features. For example, our version of the LZPiX application could be systematically extended with location-awareness and to support multiple languages as well as new run-time platforms. In particular, the mobile edition could benefit from an integration of touch-based concepts for the newer generation of smartphones. Such updates to an application could be performed either on the existing code base or directly by compiling new features to the underlying database and hence applying additional adaptations at run-time.

The two use cases have also helped us to identify best practices for XCML developers. For example, the LZPiX application based on XCML is constrained to semantically higher-level adaptations in order to avoid an 'over-componentisation' of context-aware applications and to prevent the definition of too many smaller variants. On the other hand, the Guardian example showed that existing technologies such as HTML5 and CSS3 already provide a good level of support for many client-side adaptations, e.g. based on CSS3 media queries. XCML should therefore be used to leverage and complement, rather than replace, existing technological support, e.g. to enable the specification of multimedia content in multiple variants since there was no practical solution for this using only HTML5 and CSS3.

Finally, our experiments with OpenLaszlo and HTML5/CSS3 highlight the general problem that state-of-the-art technologies and even the latest web standards still lack common concepts and methods for the specification of context and context-aware adaptations. For example, CSS3 media queries cover only a few device aspects such as screen width and height, aspect ratio or resolution. However, other device features such as supported input modalities are currently not considered in the standardisation efforts. Vendors have therefore started to address this deficiency by extending media queries with proprietary features, e.g. to be able to detect whether a device supports touch input. Moreover, the support for adaptivity based on media queries is also insufficient if required adaptations need to concern, not only the layout, but also the content of a web page or associated behaviour. While XCML provides this uniform view over different levels of adaptations, in the case of native web technologies, JavaScript would be additionally required to access web page elements and adapt associated content or behaviour programmatically. However, this is one of the cases where a bridge between technologies is required since JavaScript does not provide the methods for directly accessing device characteristics in the same manner as CSS media queries. As shown in our case study, there are workarounds, but this is also where the proposal we make with XCML can provide new starting points for more elegant solutions that could bring potential benefits to future web standards.

8 Conclusion

Our work was motivated by the increased proliferation and diversity of new devices and the varying support for context-aware constructs and mechanisms in state-of-the-art languages and technologies. We have presented XCML, a domain-specific language and framework for the development of context-aware and adaptive web applications designed to help developers cater for new adaptation scenarios in a responsive manner. The contributions of XCML comprise a context-aware language binding for XML, a finer context representation compared to previous approaches and a powerful context algebra and matching process. The essence of XCML is its support for context matching expressions that enable the definition of context-dependent variants at the different levels of content, structure and presentation.

While we have focused on XCML as a language and how it leverages the proposed principles on top of XML, the goal is not to replace existing approaches with a new language or execution environment, but to show that the concepts are technically sound and that an implementation is feasible. In that sense, XCML should serve as an example of how existing languages and technologies, that are already established and clearly widespread, could potentially benefit from our proposal. This could be enabled by the ideas and underlying formalisms used to develop the language and general support for context-awareness as presented in this paper. Our on-going research aims to widen the scope of the context-aware principles to support the adaptation to other contexts of use that were not in the focus of this paper. The work is continuously driven by a special emphasis on developer support and more lightweight methods for adaptation. One of the main goals is to inform the design and implementation as well as the required forms of adaptation by exercising different adaptation strategies, e.g. to support the unique requirements of the wide variety of touch devices nowadays also used for web access.

References

1. Baumeister, H., Knapp, A., Koch, N., Zhang, G.: Modelling adaptivity with aspects. In: Proc. ICWE, pp. 406–416 (2005)
2. Belotti, R., Decurtins, C., Grossniklaus, M., Norrie, M.C., Palinginis, A.: Interplay of content and context. J. Web Eng. **4**(1), 57–78 (2005)

3. Bolchini, C., Curino, C.A., Quintarelli, E., Schreiber, F.A., Tanca, L.: A data-oriented survey of context models. SIGMOD Rec. **36**, 19–26 (2007)
4. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A unifying reference framework for multi- target user interfaces. Interact. Comput. **15**, 289–308 (2003)
5. Casteleyn, S., Woensel, W.V., Houben, G.J.: A semantics-based aspect-oriented approach to adaptation in web engineering. In: Proc. Hypertext, pp. 189–198 (2007)
6. Ceri, S., Daniel, F., Demaldé, V., Facca, F.M.: An approach to user-behavior-aware web applications. In: Proc. ICWE, pp. 417–428 (2005)
7. Ceri, S., Daniel, F., Facca, F.M., Matera, M.: Model-driven engineering of active context-awareness. World Wide Web **10**(4), 387–413 (2007)
8. Ceri, S., Daniel, F., Matera, M., Facca, F.M.: Model-driven development of context-aware web applications. TOIT **7**(1), Article 2 (2007)
9. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann Publishers Inc. (2002)
10. Daniel, F., Pozzi, G.: An open ECA server for active applications. JDM **19**(4), 1–20 (2008)
11. Daniel, F., Matera, M., Pozzi, G.: Managing runtime adaptivity through active rules: the bellerofonte framework. JWE **7**(3), 179–199 (2008)
12. De Virgilio, R., Torlone, R.: Modeling heterogeneous context information in adaptive web based applications. In: Proc. ICWE, pp. 56–63 (2006)
13. De Virgilio, R., Torlone, R., Houben, G.J.: Rule-based adaptation of web information systems. World Wide Web **10**(4), 443–470 (2007)
14. Fiala, Z., Hinz, M., Meißner, K., Wehner, F.: A component-based approach for adaptive, dynamic web documents. JWE **2**(1&2), 58–73 (2003)
15. Fiala, Z., Frăsincar, F., Hinz, M., Houben, G.J., Barna, P., Meißner, K.: Engineering the presentation layer of adaptable web information systems. In: Proc. ICWE (2004)
16. Frăsincar, F., Houben, G.J., Barna, P.: HPG: the Hera presentation generator. JWE **5**(2), 175–200 (2006)
17. Frăsincar, F., Houben, G.J., Barna, P.: Hypermedia presentation generation in Hera. IS **35**(1), 23–55 (2010)
18. Grossniklaus, M., Norrie, M.C.: An object-oriented version model for context-aware data management. In: Proc. WISE, pp. 398–409 (2007)
19. Grossniklaus, M., Norrie, M.C.: Supporting different patterns of interaction through context-aware data management. JWE **7**(3), 200–219 (2008)
20. Helms, J., Abrams, M.: Retrospective on UI description languages, based on eight years' experience with the User Interface Markup Language (UIML). IJWET **4**(2), 138–162 (2008)
21. Hennicker, R., Koch, N.: A UML-based methodology for hypermedia design. In: Proc. UML, pp. 410–424 (2000)
22. Houben, G.J., Barna, P., Frăsincar, F., Vdovják, R.: Hera: development of semantic web information systems. In: Proc. ICWE, pp. 529–538 (2003)
23. Kappel, G., Retschitzegger, W., Schwinger, W.: Modeling customizable web applications - a requirement's perspective. In: Proc. ICDL, pp. 168–179 (2000)
24. Katz, R.H.: Toward a unified framework for version modeling in engineering databases. ACM Comput. Surv. **22**, 375–409 (1990)
25. Koch, N., Wirsing, M.: The Munich reference model for adaptive hypermedia applications. In: Proc. AH, pp. 213–222 (2002)
26. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López-Jaquero, V.: UsiXML: UsiXML: a language supporting multi-path development of user interfaces. In: Proc. EHCI/DS-VIS, pp. 200–220 (2004)
27. Nebeling, M., Grossniklaus, M., Leone, S., Norrie, M.C.: Domain-specific language for context-aware web applications. In: Proc. WISE, pp. 471–479 (2010)
28. Nebeling, M., Matulic, F., Norrie, M.C.: Metrics for the evaluation of news site content layout in large-screen contexts. In: Proc. CHI, pp. 1511–1520 (2011)
29. Nebeling, M., Matulic, F., Streit, L., Norrie, M.C.: Adaptive layout template for effective web content presentation in large-screen contexts. In: Proc. DocEng (2011)
30. Niederhausen, M., Karol, S., Aßmann, U., Meißner, K.: HyperAdapt: enabling aspects for XML. In: Proc. ICWE, pp. 461–464 (2009)
31. Niederhausen, M., van der Sluijs, K., Hidders, J., Leonardi, E., Houben, G.J., Meißner, K.: Harnessing the power of semantics-based, aspect-oriented adaptation for AMACONT. In: Proc. ICWE, pp. 106–120 (2009)

32. Norrie, M.C., Signer, B., Grossniklaus, M., Belotti, R., Decurtins, C., Weibel, N.: Context-aware platform for mobile data management. Wirel. Netw. **13**, 855–870 (2007)
33. Paternò, F., Santoro, C., Mäntyjärvi, J., Mori, G., Sansone, S.: Authoring pervasive multimodal user interfaces. IJWET **4**(2), 235–261 (2008)
34. Paternò, F., Santoro, C., Spano, L.: MARIA: a universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. TOCHI **16**(4), 19:1–19:30 (2009)
35. Stavrakas, Y., Gergatsoulis, M.: Multidimensional Semistructured Data: Representing context-dependent information on the web. In: Proc. CAiSE, pp. 183–199 (2002)
36. Wadge, W.W., Brown, G., m.c. schraefel, Yildirim, T.: Intensional HTML. In: Proc. PODDP, pp. 128–139 (1998)