Ersch. in: Web information systems engineering : 11th international conference ; proceedings / WISE 2010, Hong Kong, China, December 12 - 14, 2010. Lei Chen, Peter Triantafillou, Torsten Suel (eds.). - Berlin [u.a.] : Springer, 2010. - S. 471-479. - (Lecture notes in computer science ; 6488). - ISBN 978-3-642-17615-9

http://dx.doi.org/10.1007/978-3-642-17616-6_42

Domain-specific Language for Context-aware Web Applications

Michael Nebeling, Michael Grossniklaus,
Stefania Leone, and Moira C. Norrie

Institute of Information Systems, ETH Zurich
CH-8092 Zurich, Switzerland
`{nebeling|grossniklaus|leone|norrie}@inf.ethz.ch`

Abstract. Context-awareness is a requirement in many modern web applications. While most model-driven web engineering approaches have been extended with support for adaptivity, state-of-the-art development platforms generally provide only limited means for the specification of adaptation and often completely lack a notion of context. We propose a domain-specific language for context-aware web applications that builds on a simple context model and powerful context matching expressions.

1 Introduction

Context-awareness has been recognised as an important requirement in several application domains ranging from mobile and pervasive computing to web engineering. Particularly in web engineering, it has been proposed to address personalisation, internationalisation and multi-channel content delivery. With clients ranging from desktop computers, laptops and mobile devices to digital TVs and interactive tabletops or wall displays, web applications today have to support many different device contexts and input modalities.

Researchers have mainly addressed the need for adaptivity in web appliations on the conceptual level by adding support for specifying context-sensitive behaviour at design-time into existing methodologies. Most of these model-driven approaches also feature code generators capable of deploying adaptive web applications, but often lack comprehensive run-time support for managing context-aware sites. Practitioners, on the other hand, tend to build on development platforms such as Silverlight, JavaFX and OpenLaszlo to cater for multiple run-time platforms; however, the methods and languages underlying these frameworks often follow a less systematic approach and, in particular, lack support for context-aware constructs and mechanisms.

In this paper, we propose a domain-specific language for context-aware web applications that extends and consolidates existing approaches based on context-aware concepts in markup languages. We present a possible execution environment that builds on a database-driven system developed in previous work [1]. We begin in Sect. 2 by discussing related work. Our approach is described in Sect. 3. Section 4 presents an implementation of the proposed language followed by concluding remarks in Sect. 5.

2 Background

Various model-based approaches have addressed the need for adaptivity in web appliations. For example, WebML [2] has introduced new primitives that allow the behaviour of context-aware and adaptive sites to be modelled at design-time [3]. In UWE [4], the Munich Reference Model [5] for adaptive hypermedia applications provides support for rule-based adaptation expressed in terms of the Object Constraint Language (OCL). The specification of adaptive behaviour is an integral part of the Hera methodology [6], where all design artefacts can be annotated with appearance conditions. These annotations will then be used to configure a run-time environment such as AMACONT [7] which supports run-time adaptation based on a layered component-based XML document format [8]. Finally, the Bellerofonte framework [9] is a more recent approach using the Event-Condition-Action (ECA) paradigm. The implementation separates context-aware rules from application execution by means of a decoupled rule engine with the goal of supporting adaptivity at both design and run-time.

In a second stream, research on user interface description languages has spawned a number of different models and methods with the most prominent example being the CAMELEON framework [10] which separates out different levels of user interface abstraction to be able to adapt to different user, platform and environment contexts. A diversity of languages and tools have been proposed (UsiXML [11], MARIA [12], etc.), and W3C is forming a working group on future standards of model-based user interfaces.[1]

State-of-the-art development practices show a significant gap between the methodical approaches at the conceptual level and existing implementation platforms at the technological level. For example, while researchers promote the importance of design processes such as task modelling, in practice, this step is often skipped and developers instead use visual tools such as Photoshop and Flash for rapid prototyping and to directly produce the concrete user interfaces. Moreover, even though platforms such as Silverlight, JavaFX and OpenLaszlo have been specifically designed to support multiple run-time environments, the underlying languages (Microsoft XAML, JavaFX Script, OpenLaszlo LZX, etc.) use many different approaches and lack common concepts and vocabulary, let alone a unified method, for the specification of context-aware adaptation.

Early research on Intensional HTML (IHTML) [13] demonstrated how context-aware concepts can be integrated into HTML with the goal of declaring versions of web content inside the same document. The concepts used in IHTML were later generalised to form the basis for Multi-dimensional XML (MXML), which in turn provided the foundation for Multi-dimensional Semi-structured Data (MSSD) [14]. However, these languages still suffer from two major design issues. First, the multi-dimensional concepts are not tightly integrated with the markup and need to be specified in separate blocks of proprietary syntax. This renders existing HTML/XML validators and related developer tools void as any attempt to parse IHTML/MXML documents will fail. Second, it becomes ex-

---

[1] `http://www.w3.org/2010/02/mbui/program.html`

tremely difficult to manage such versioned documents, at least for a considerable number of contextual states, if the possible states are not clearly defined in some sort of context model.

3 Approach

To alleviate the aforementioned problems with IHTML and MXML, we designed XCML not to support versioning of single web documents, but instead to augment entire web sites with context-dependent behaviour based on the following three core principles.

- **XML-based markup language**. We base XCML on XML since many web development frameworks build on XML-based markup languages. XCML introduces a set of proprietary, but well-defined, tags that integrate context-aware concepts into the markup while still producing well-formed XML.
- **Application-specific context model**. Context-aware web sites based on XCML must first define the context dimensions and possible states specific to the respective application. To facilitate the management of larger version spaces, we enforce a clear separation between context state definitions in the header and context matching expressions in the body of XCML documents.
- **Context algebra and expression language**. A distinguishing feature of our approach is that context is a refining concept used to augment rather than completely specify content delivery. XCML is hence based on context *matching* expressions as opposed to statements formulated in an *if-then-else* manner. Instead of relying on rather strict ECA rules as used in [9, 5], XCML requests trigger a matching process in which the best-matching variants are determined based on scoring and weighting functions as defined in [1].

To keep the specification of context models simple, we use a very general context representation based on the notion of context dimensions and states. A *context dimension* represents a set of characteristics or semantically grouped factors to be taken into account when compiling the context-aware web site, while a *context state* describes a valid allocation of such a dimension. The following example defines a simple context model to distinguish different browser contexts.

```
<xcml:context>
 <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="3..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..8" />
 </xcml:context-state>
 <xcml:context-state name="Firefox" dimension="Browser">
  <xcml:context-property key="agent" value="Firefox" />
  <xcml:context-property key="version" value="1..3.8" />
 </xcml:context-state>
 [..] <-- other dimensions and states -->
</xcml:context>
```

The extract shows the definition of the Browser dimension in terms of agent and version. With the states defined thereafter, an application can distinguish between previous and current versions of Internet Explorer and Firefox. The potential advantages of our approach are clearer when looking at context models with multiple dimensions. For example, in a three-dimensional context space, an XCML body could be defined to distinguish not only browsers but also languages English and German as well as desktop and mobile platforms as follows.

```
<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>
    <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>
   </head>
   <body> <xcml:component select="*" />[..]</body>
  </html>
 </xcml:layout-variant>
 <xcml:layout-variant match="Mobile"> [..] </xcml:layout-variant>
</xcml:layout>
<xcml:component name="website">
 <xcml:component-variant layout="websiteLayout" match="Desktop">
  <xcml:child-components>
   <xcml:component name="header">
    <xcml:component-variant type="headerType"  match="English">
     <xcml:attribute-value name="title" value="XCML Site" /> [..]
    </xcml:component-variant>
    <xcml:component-variant match="German">[..]</xcml:component-variant>
   </xcml:component>
   [..] <!-- other components -->
  </xcml:child-components>
 </xcml:component-variant>
 <xcml:component-variant match="Mobile"> [..] </xcml:layout-variant>
</xcml:component>
```

As with many existing approaches, we separate presentation concepts from content and navigation. In XCML, we achieve this by means of a simple component-based model that allows for versioning on each layer in terms of so-called variants. The element websiteLayout in the example above defines variants to place the web site's title in the best-matching language, link stylesheets optimised for Internet Explorer or Firefox and recursively include all children of the associated components. The website component associated with the layout defines content variants such as the title in English and German and structure variants for the desktop and mobile platforms.

XCML supports versioning at the data, structure and layout levels for maximum flexibility. The essence of XCML is however the underlying context algebra that allows for the specification of context-dependent behaviour along simple but powerful context matching expressions. The match clause "Firefox" used in the example directly translates to $\{(agent, Firefox), (version, 1..3.8)\}$ and, in this case, constitutes a best-match for all Firefox browsers versions 1 to 3.8. XCML also supports more complex context expressions such as "old_IE or new_IE" and "not Firefox" or even those that span multiple dimensions as in "Desktop and (old_IE or new_IE)". The evaluation of such expressions involves two steps. First, each context state will be substituted with the respective context properties. This would mean that the latter expression evaluates to "{(type,desktop)} and ({(agent,IE), (version,7)} or {(agent,IE), (version,8)})". A dimensional analysis would then find that the left part of the given expression is of the device dimension (which defines the key type) while the right part translates to the browser dimension (which defines the two keys agent and version) for both the left and right operand. In the second step, all propositional connectives will be resolved to give {(type,desktop)} and {(agent,IE), (version,7..8)} and finally to {(type,desktop), (agent,IE), (version,7..8)}.

After evaluating each context expression, it is necessary to compare the resulting context states against the context for which the web site is compiled. Only if equal or at least a partial match will a variant associated with a context state become part of the whole version. In our example, this would mean that the variant associated with {(type,desktop), (agent,IE), (version,7..8)} would only be displayed if the web site is accessed from the desktop using either Internet Explorer 7 or 8 (independent of the selected language).

4 Implementation

We have implemented an execution environment for XCML based on a generalisation of the XCM system that we developed in previous work [1]. By mapping the XCML language concepts to the previously established concepts for context-aware data management, we can use the database as a cache for the context-dependent variants as shown in Fig. 1. We have defined several components for the processing of XCML, which now form the new *design-time* and complement the existing *run-time* of the XCM system.

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.

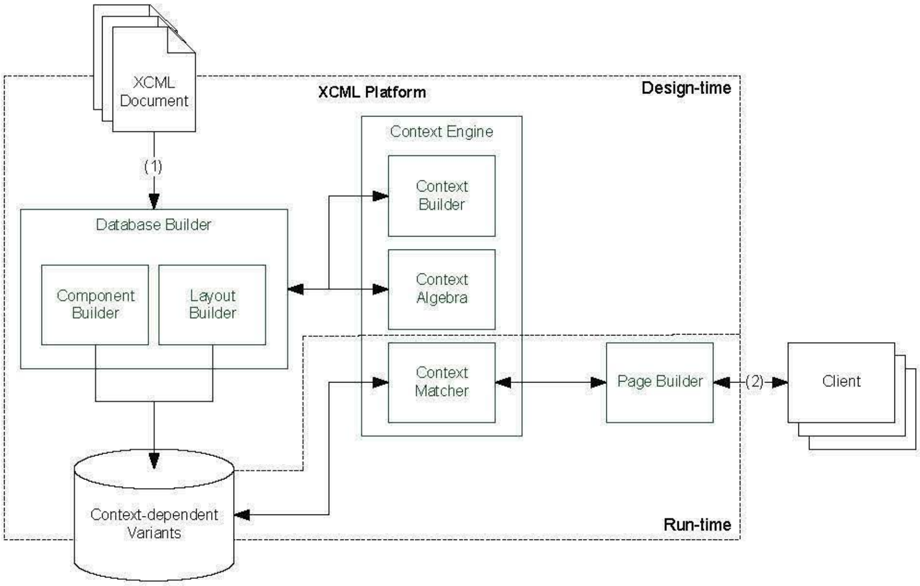

Fig. 1: Integration of the new design-time components with XCM run-time

- *Database Builder.* As the Database Builder parses the XCML documents, it employs the Context Builder to build the context space and creates the corresponding context-dependent variants in the database.
- *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 that implements the two-step evaluation process of context expressions described in Sect. 3. The role of the Context Matcher is to score all variants of a specific object and determine the best-matching version for a given context at run-time as described in detail in [1].
- *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. Again, details are given [1].

We now explain how XCML documents are processed and compiled into the database (process (1) in Fig. 1). As shown in Fig. 2, presentation instructions within `xcml:layout` tags are handled differently from XCML component variants.

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.

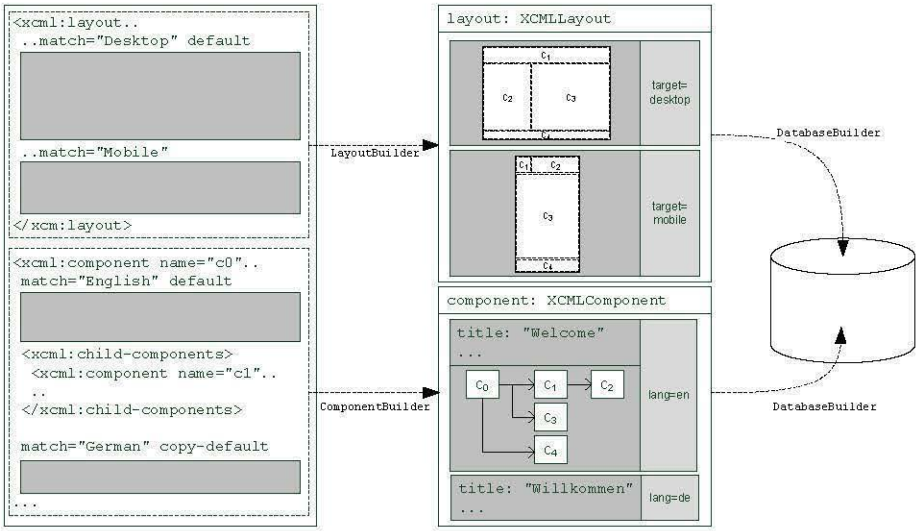

Fig. 2: Processing of XCML layouts and components

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 reuse are somewhat 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. If the 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.

Layouts. For each `xcml:layout` defined in an XCML document, the Layout Builder preprocesses 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 the 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.

5 Conclusion

We have presented a domain-specific language that promotes a simple context model and powerful context matching expressions. Our work was motivated by the lack of context-aware concepts in state-of-the-art web development platforms and languages. In future work, we want to improve the integration with existing frameworks and also design adequate tool support for the development and debugging of adaptive web sites based on our language.

References

1. Grossniklaus, M., Norrie, M.C.: An Object-Oriented Version Model for Context-Aware Data Management. In: Proc. WISE. (2007) 398–409
2. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann Publishers Inc. (2002)
3. Ceri, S., Daniel, F., Matera, M., Facca, F.M.: Model-driven Development of Context-Aware Web Applications. TOIT **7**(1) (2007) Article 2
4. Hennicker, R., Koch, N.: A UML-Based Methodology for Hypermedia Design. In: Proc. UML. (2000) 410–424
5. Koch, N., Wirsing, M.: The Munich Reference Model for Adaptive Hypermedia Applications. In: Proc. AH. (2002) 213–222
6. Houben, G.J., Barna, P., Frăsincar, F., Vdovják, R.: Hera: Development of Semantic Web Information Systems. In: Proc. ICWE. (2003) 529–538
7. Fiala, Z., Hinz, M., Houben, G.J., Frăsincar, F.: Design and Implementation of Component-based Adaptive Web Presentations. In: Proc. SAC. (2004) 1698–1704
8. Fiala, Z., Hinz, M., Meissner, K., Wehner, F.: A Component-based Approach for Adaptive, Dynamic Web Documents. JWE **2**(1-2) (2003) 58–73
9. Daniel, F., Matera, M., Pozzi, G.: Managing Runtime Adaptivity through Active Rules: the Bellerofonte Framework. JWE **7**(3) (2008) 179–199
10. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi- Target User Interfaces. Interacting with Computers **15**(3) (2003) 289–308
11. 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. (2005) 200–220
12. Paternò, F., Santoro, C., Spano, L.: MARIA: A Universal Language for Service-Oriented Applications in Ubiquitous Environment. ACM Trans. on Computer-Human Interaction **16**(4) (2009) Article 19
13. Wadge, W.W., Brown, G., m. c. schraefel, Yildirim, T.: Intensional HTML. In: Proc. PODDP. (1998) 128–139
14. Stavrakas, Y., Gergatsoulis, M.: Multidimensional Semistructured Data: Representing Context-Dependent Information on the Web. In: Proc. CAiSE. (2002) 183–199