Modeling Ubiquitous Web Applications - The WUML Approach

Gerti Kappel [1], B. Pröll[2], Werner Retschitzegger [1], Wieland Schwinger [3]

[1] Institute of Applied Computer Science, Department of Information Systems (IFS)
University of Linz, Altenbergerstraße 69, A-4040 Linz, Austria
{gerti | werner}@ifs.uni-linz.ac.at

[2] Institute for Applied Knowledge Processing (FAW)
Softwarepark Hagenberg, Hauptstraße 99, A-4232 Hagenberg, Austria
bproell@faw.uni-linz.ac.at

[3] Software Competence Center Hagenberg (SCCH)
Softwarepark Hagenberg, Hauptstraße 99, A-4232 Hagenberg, Austria
wieland.schwinger@scch.at

Abstract. E-commerce and m-commerce have dramatically boosted the demand for services which enable ubiquitous access. Ubiquity with its anytime/anywhere/anymedia nature requiring context-aware computing calls for new engineering techniques supporting these kind of services. In this paper, we propose the notion of customisation as the uniform mechanism to deliver ubiquitous web applications providing adaptability with respect to a certain context. As a prerequisite for supporting customisation design, a set of generic models is introduced comprising a context model, a profile model, and a rule model. At the application's side, customisation hooks are provided representing the major hot spots of adaptation. A customisation toolkit in terms of a customisation rule editor and browser supports an integrated modelling process and facilitates reusability on the basis of a repository of customisation rules and patterns.

1 Introduction

The Internet and in particular the World Wide Web have introduced a new era of computing, providing the basis for promising application areas like e-commerce and m-commerce [10], [19]. These application areas have dramatically boosted the demand for services which enable *ubiquitous access*, being mainly characterised by the *anytime/anywhere/anymedia paradigm* [23]. Considering ubiquitous web applications from a software engineering point of view, as their complexity increases, so does the importance of modelling techniques [1], [3], [7]. Models of a ubiquitous web application prior to its construction are essential for comprehension in its entirety, for communication among project teams, and to assure architectural soundness and maintainability. There are already a couple of methods especially dedicated to the modelling of web applications, which however lack proper support for the issues arising when dealing with ubiquity [12].

We propose that a ubiquitous web application should be designed from the start taking into account not only its hypermedia nature, but also the fact that it must run "as is" on a variety of platforms, including mobile phones, Personal Digital Assistants (PDAs), full-fledged desktop computers, and so on. This implies that a ubiquitous

web application has to take into account the different capabilities of devices comprising display size, local storage size, method of input and computing speed as well as network capacity. New opportunities are offered in terms of location-based, time-based, and personalised services taking into account the needs and preferences of particular users. Consequently, a ubiquitous web application must be *context-aware*, i.e., aware of the environment it is running in.

This paper establishes the notion of *customisation* as uniform mechanism to provide adaptability of a ubiquitous web application with respect to a certain context. In particular, the contribution can be summarised as follows:

- We give an overview on the state of the art of *ubiquitous web application engineering* and highlight the *different facets of customisation* from a historical point of view.
- We give insight into appropriate *customisation modelling concepts* and *notations* by proposing a *generic customisation model* which comprises a *context model*, a *profile model* and a *rule model* and by separating the application into a *stable part* and a *variable part* which provides appropriate *customisation hooks.*
- We outline first ideas about proper tool support for customisation design in terms of a *customisation toolkit* which provides a *customisation rule designer* and *browser* and we propose the abstraction of concrete customisation rules into *customisation rule patterns* which are part of a *pattern library* to facilitate reusability.

2 Customisation - the Notion and the Issues

Considering the notion of customisation from a historical point of view, it represents a major challenge at least since the end user has been put in the middle of concern when developing interactive applications. An area dealing with customisation issues already for a long time is the user interface community, which brought up the notion of *adaptive user interfaces,* cf., e.g., [9]. Adaptive user interfaces are designed to tailor a system's interactive behaviour considering both individual needs of human users and changing conditions within an application environment. The broader approach of *intelligent or advisory user interfaces* includes adaptive characteristics as a major source of its intelligent behaviour, cf., e.g., [5]. Another area dealing with customisation but emphasising more on adapting the content of an application are *information filtering* and *recommender systems* [2]. The goal of these systems is to go through large volumes of dynamically generated textual information and present to the user those which are likely to satisfy his/her information requirements. With the emerge of hypertext the need for alternative access paths to information in terms of, e.g., different navigation structures became prevalent leading to another research direction called *adaptive hypertext and hypermedia* [4]. Last but not least, the proliferation of *mobile computing* and *mobile web applications*, in particular, makes it necessary to consider not only the user preferences but also the environment in terms of, e.g., location information or device characteristics in order to adapt the application properly [17].

Customisation Dimensions. Learning from these different areas, the *design space of customisation* can be characterised by three orthogonal dimensions, comprising the *degree of customisability*, the *granularity of adaptation* and the *kind of context* (cf. Fig. 1) [15].

Degree of Customisability. The first dimension, the *degree of customisability*, expresses that both, context and adaptation can be either *static*, i.e., pre-defined or *dynamic*, i.e., determined during run-time. An example for static context and adaptation could be to select a pre-defined version of a certain application depending on the device used. An example for the fully dynamic case would be to adapt the resolution of an image on the fly, due to a change in bandwidth. Applications supporting only static contexts and/or adaptations are often called *adaptable* whereas those providing also dynamic concepts are considered to be *adaptive* [16].

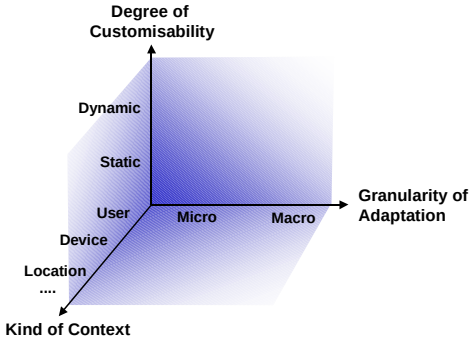

Fig. 1. Dimensions of Customisation

Granularity of Adaptation. The second design dimension indicates the *granularity of adaptation* ranging from *micro adaptation* to *macro adaptation*. Whereas micro customisation is concerned with fine-grained adaptations (e.g., disabling a link on a page), macro adaptation means that depending on the context, rather large parts of an application are adapted (e.g., instead of an indexed guided tour, use a simple bullet list). Note that, there is no exact border between micro and macro adaptation. In its most extreme form, macro adaptation simply means that depending on the context, the whole application realising a certain service is substituted by another one, thus better fitting in the current context. This extreme form of macro customisation often occurs together with the combination of static context and adaptation. As encountered in [15], the few existing approaches dealing with modelling aspects of ubiquitous web applications adhere in large parts to this kind of adaptation, e.g., by modelling several so-called "site views" each one tailored to another context.

Kind of Context. The third dimension covers the *kind of context* which is considered by customisation. In this respect, it is useful, to consider existing approaches to customisation as done in [12]. The majority of these approaches consider the issue of *personalisation* by making assumptions about relevant user characteristics and preferences to get personalised services from a certain resource, or even to personalise already the discovery of resources. Also a considerable number of approaches take *network and device properties* into account. Those two are often considered together which is reasonable since mobile devices also imply a wireless connection carrying certain network constraints. In [12] it has been encountered, however, that only few of the surveyed approaches explicitly regard location information. This is due to both technical deficiencies and lack of legal regulations.

Semantic Enhancement vs. Semantic Equivalence. Those approaches supporting personalisation or providing location information endow the application with *semantic enhancement*, in that each particular user is provided with specific added value. Such enhancement actually makes the same application provide increased value for different users, who ultimately perceive the application as two different services. On the other hand, the same application customised for the same user may (and certainly does) look different when it is run on different devices and/or in different situations. This is inevitable (for example it is impossible to show that beautiful applet on a PDA with no virtual machine installed), but the service (or the added value) provided to the user should nevertheless be the same. In this case customisation enables to maintain *semantic equivalence*. In this case we talk of semantic equivalence, which means that, despite the different context, the value provided to the user should still be the same.

3 The WUML Approach to Customisation Design

WUML stands for *Web Unified Modeling Language* and aims at the methodological support of web application development with special focus on ubiquity. The design goals behind WUML are

- usage of *UML* [22] as the basic formalism,
- web-tailored extension of UML using the *UML extension mechanisms* only,
- specification of WUML in terms of a *generic framework*,
- *three-level view* on the development process (cf. [21]) taking *content* (i.e., domain-dependent data) *hyperbase* (i.e. the logical composition of web pages together with the navigation structure), and *presentation* (i.e., the layout of each page as well as user interaction) into account, and
- support of *ubiquity* through *customisation*.

In this paper, we focus on the customisation model of WUML, for all other aspects we refer to [14].

Our approach to customisation design is based on a broad view on customisation in that it takes into account all three dimensions of customisation as identified in the previous section. First, although most often separated in existing approaches, we think that customisation for ubiquitous web applications should uniformly consider *personalisation* aspects, together with issues resulting from the *anywhere/anytime/anymedia paradigm*, herewith providing both, semantic enhancement and semantic equivalence. Second, customisation design should not only consider *macro adaptation* but also *more fine-grained forms of adaptation* in order to be able to better reflect a certain context when tailoring an application (again with respect to semantic enhancement and semantic equivalence). Third, since in our opinion, static context and static adaptation does not reflect the dynamic nature of ubiquitous web applications, a special focus of customisation design is given on dynamic context and dynamic adaptation.

Overall Architecture. The architecture given in Fig. 2 provides the basis for accomplishing such a broad view of customisation. The *context* together with *profiles* provide detailed information about the environment of an application and trigger the actual customisation as soon as the environment changes. Whereas context represents current and historical information about the environment of the application which is

automatically monitored, profiles cover more stable information which are explicitly given by a designer or a user, e.g., user preferences or the vendor of a device, e.g., descriptions of device properties. A rule-based mechanism in terms of *customisation rules* is employed in order to specify the actual customisation. Customisation rules monitor changes in context, profile and application state itself. For separation of concerns, the application is separated into a *stable part*, comprising the default, i.e., context-independent structure and behaviour of the application and a *variable, context-dependent part*, thus being subject to most of the adaptations.

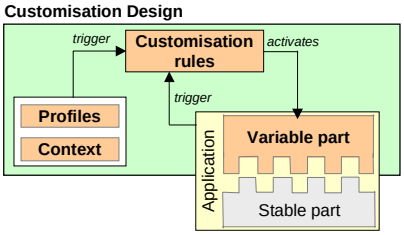

Fig. 2. Overall Architecture of the WUML Approach to Customisation Design

Customisation Design vs. Traditional Application Design. Finally, it has to be noted that customisation could also be made part of traditional application design without explicitly considering customisation rules, profiles, context and the variable part of an application separately from its stable part. By means of factoring out these aspects from the application already during the early stages of the software lifecycle, however, the dynamic nature of ubiquitous web applications can be much better dealt with, not least with respect to reusability and locality of changes. The rationale behind an explicit customisation design is similar to the motivation given for factoring out business policies from object-oriented applications [13].

4 Customisation Model

To support the architecture introduced in the previous section, we propose a *generic customisation model* in the sense of an object-oriented framework. Generic means in this respect that the model is domain-independent and provides a couple of pre-defined classes which can be used for customising a certain web application. In addition, the pre-defined classes can be extended by the designer through subclassing in order to cope with application specific details. The provided generic models are discussed in more detail in the forthcoming subsections.

4.1 Context Model

As already mentioned, we define context as the *reification of the environment in terms of its properties* which are continuously, dynamically updated to take into account the fact that the environment also continuously changes. For us the context is *not modifiable* and so is outside of the control of the ubiquitous web application. It just provides a manageable description of the environment so that it can be addressed within customisation design. Although the context is *application independent*, the

designer needs to specify which properties of the environment are of interest for a particular application.

Context Properties. Based on the existing literature discussed in the previous section and considering what is currently made available by existing technology, our generic context model considers the following *context properties* which can be, as already mentioned, extended by the designer:

- *User Agent:* The user agent property refers to the demand of ubiquitous web applications for anymedia. It provides information about the device and browser. Together with a device profile and a browser profile (cf. Section 4.2) this allows to identify constraints relevant for a proper adaptation of the application.
- *User:* The knowledge about the user takes into account the necessity of personalisation. Looking at existing technology, the provided telephone number together with user profile information (cf. Section 4.2) allows identifying the individual user and user class.
- *Network:* Context properties concerning the network comprise, e.g., the bandwidth.
- *Location:* Location copes with the need for mobile computing and captures information about the location an application is accessed. Actually, this information is not directly provided by mobile devices themselves but is obtained via a so-called location server.
- *Time:* Finally, the context property time allows to customise the application with respect to certain timing constraints such as opening hours of shops or timetables of public transportation. Currently the time context property is represented at the server only.

Session, Current Context and History. Since web applications enforce the notion of sessions, possibly consisting of a sequence of transactions, these context properties need to be considered within the boundaries of sessions, i.e., each session has its own context. Furthermore, since the context within a session is subject to continuous changes, it is necessary to identify the most recent reification of the environment, which is further on called *current context*, using the latest timestamp. The current context comprises the current values of the context properties for a given interaction (e.g., the most recent) within the session of a ubiquitous web application. Practice has shown that it is necessary to broaden the view on context in that a context does not only consider the current context at a given point in time but also historical information. This is necessary to be able to identify changes in the values of the context properties over time. Thus the context model also needs to include a *history dimension*, in that a relevant context `C` can be formulated as a vector of context property values over time. For example, to determine user navigation patterns or the average throughput of a system, it is necessary to collect information about historic interactions. In contrast, the information about which device is used allowing customising the presentation to fit to a restricted display size mostly requires information of the current device only.

Our understanding of an appropriate context model is shown in Fig. 3 by means of a UML class diagram. Since every context information needs to be considered within a session the class `Session` acts as its root and at the same time holds the collection of all session instances within the system. For each session the class `History` can hold a sequence of contexts indexed by timestamps. The context, represented by the class `Context`, is an aggregation of a number of context properties, each of which is a subclass of the abstract `ContextProperty` class. Note that the `Context` class is

not an aggregation of the generic abstract class but rather of the specific, concrete subclasses.

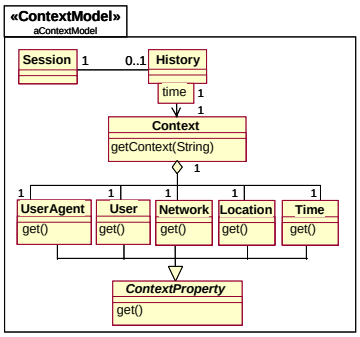

Fig. 3. Context Model

We chose this representation because we prefer to have a *context vector* that explicitly represents the specific subclasses herewith emphasising the fact that at a certain point in time only one instance of each specific context subclass may be present.

4.2 Profile Model

Profiles, in contrast to the context, represent more stable information which is explicitly given by, e.g., a user, a designer or the vendor of a device. The proper representation of profile information is already subject to standardisation efforts. The World Wide Web Consortium (W3C) is working on a framework for the management of device and user profile information called *"Composite Capabilities / Preference Profiles" (CC/PP)* which is based on the Resource Description Framework (RDF) [24]. It specifies how client devices express their capabilities and users express their preferences to the web application server. In addition, it defines recommendations about the content of such profiles. One major goal followed by CC/PP is extensibility so that new properties can be defined and included in the description and users can overwrite vendor-defined default preferences.

We build on this standardisation effort when modelling profile information. Depending on the kind of profile, information may be available already at *design time, e.g.* device characteristics, or, if the profile is application dependent, not before *run-time*, e.g. usage statistics. In the latter case, the designer of the ubiquitous web application needs to take care to explicitly provide the information throughout the application run-time. For this the profile information needs to be considered as any other information.

Analogous to the context model described in the previous section, our profile model encompasses the following profiles:

- *User Agent:* User agent profiles describe both, the capabilities of devices, e.g., display size, memory, operating system and the browsers running on these devices, e.g., kind of browser and version number. User agent profiles are application independent.

- *User:* A user profile can comprise information that is *voluntarily entered* by the user describing the user's preferences with respect to application customisation as well as information that is *transparently acquired* by the system including, e.g., usage statistics. Note that in general the user profile is application dependent as opposed to the user agent profile and the context.
- *Network:* A network profile is application independent and could, e.g., contain maximal bandwidths of certain connection types.
- *Location:* An example for a location profile is a road map which is application independent.
- *Time:* Finally, a time profile, which is again application independent, could encompass time zones or time-of-day settings.

Similar to our generic context model, the profile model is not comprehensive but can be easily extended by the designer to represent additional kinds of profiles necessary for a particular ubiquitous web application.

4.3 Rule Model

For specifying a certain customisation we propose the usage of rules (furtheron called *customisation rules*) in terms of the *event/condition/action (ECA) mechanism*. ECA rules are well known in the area of active database systems [11] and represent among others a commonly accepted mechanism for a localised specification of business policies as opposed to spreading it over several applications [13]. Although it would be possible to use CA-rules only, we stick to an event-driven specification to better reflect the dynamic nature of ubiquitous web applications. In our approach, the event together with the condition describes the *situation* when customisation is necessary.

Event. The event part of a customisation rule determines the events which are able to trigger the rule. With respect to our overall architecture given in Fig. 2, it detects changes in context, profile or application state and therefore indicates the need for a potential customisation. An example for an event would be a change of bandwidth. For more details it is referred to Section 4.4.

Condition. The condition part is evaluated as soon as the rule is triggered by an appropriate event and evaluates whether and which adaptation is actually desired. An example would be to test whether the bandwidth falls below a certain minimum. Conditions are in fact predicates which can be combined by means of logical operators currently using OCL-syntax [22].

Action. The action part of a customisation rule is responsible for activating a certain adaptation of the application, e.g., switch to text mode. An action mainly deals with so called *customisation hooks* as provided by the designer of the ubiquitous web application (cf. Section 5) but can also activate other operations of the stable part of the application which are not explicitly foreseen for customisation purposes.

Rule Properties. Several properties of a rule, such as *priority*, *activation state* and *transaction mode* [11], may be used to further specify the actual customisation process at runtime. This in turn determines when the events are detected, when the condition is evaluated, and when the action is executed.

4.4 Event Model

Applying the ECA mechanism in the domain of ubiquitous web applications requires a dedicated event model The events of this event model need to monitor changes within both, context and profile information as well as changes in the application state, resulting from explicit user requests. Consequently, our event model considers three basic types of *primitive events* which in turn can form *composite events*. Fig. 4 shows a UML class diagram of our event model. Since we consider events as first-class objects, the application developer is able to extend the pre-defined event types by means of subclassing.

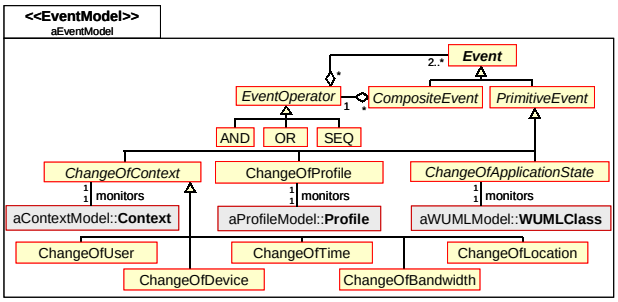

Fig. 4. Event Model

The event model provides the following three basic types of primitive events:

- *Change of context*: We propose the following pre-defined events indicating changes in the context, namely, `ChangeOfUser`, `ChangeOfDevice`, `ChangeOfBandwidth`, `ChangeOfLocation`, `ChangeOfTime`. Note, that these events directly monitor changes within the corresponding context properties of our context model.
- *Change of profile*: Since the properties of certain profiles are not necessarily limited, we propose at a first attempt a single pre-defined event only, namely `ChangeOfProfile`.
- *Change of application state*: Changes in the application state, resulting, e.g., from a certain user interaction may signal events, which are useful for customisation purposes. There exists a number of pre-defined event classes which are specific to the modelling elements of WUML and therefore not depicted in Fig. 4.

To be able to model complex real-world situations, which should be monitored, we suggest the notion of so-called *composite events*. Composite events are constructed by means of the logical event operators `AND`, `OR`, and `SEQ`, using the above mentioned primitive events or again composite events as operands. For example, the requirement that a page should be both personalised and tailored to a certain device demands for a composite event like `(ChangeOfUser OR ChangeOfDevice)`. Composite events also allow to express if the actual adaptation should be done in advance, i.e., independent of any user's request or on the fly, meaning that adaptation is not done before it is needed in response to a user's request. In the first case, customisation rules monitor changes in context, profile or application state only, whereas in the latter, the request of a user has to be taken into account too.

5 Customisation Notation

5.1 Notation for Specifying Customisation Hooks

As a major prerequisite for realising customisation we propose that the application should provide appropriate *customisation hooks*, herewith again resembling a basic idea of object-oriented frameworks. These customisation hooks provide the basis for customisation rules to adapt the application towards a particular context. As already mentioned, the application is separated into a *stable part* that provides the basic functionality, and a *variable part* that provides the customisation hooks for both, micro customisation and macro customisation. Whereas we consider micro customisation at a very fine-grained level, meaning that a *single modelling element is adapted* only, macro customisation which is based in turn on micro customisation (cf. below) covers the adaptation of *more than one modelling element*.

Micro Customisation. For modelling micro customisation, we propose that the UML model element `class`, representing a basic modelling element in WUML should be extended with two additional compartments for representing the variable part of the application in terms of customisation hooks (cf. Fig. 5).

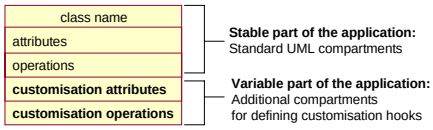

Fig. 5. Additional Class Compartments for Specifying Customisation Hooks

Macro Customisation. Concerning macro customisation, we propose to use the *UML package mechanism* in order to group those modelling elements which are subject to the same adaptation within so called *customisable packages* (using the stereotype «`CustomisablePackage`», cf. Fig. 6). Since packages allow to group an arbitrary number of arbitrary UML modelling elements together, the granularity of adaptation is not limited. Similar to classes providing customisation hooks for micro customisation within additional compartments, customisable packages provide customisation hooks within a pre-defined so called *customiser interface* (using the stereotype «`CustomiserInterface`», cf. Fig. 6).

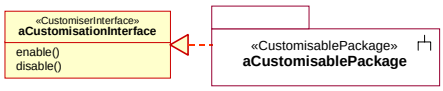

Fig. 6. Customiser Interface and Customisable Package

This customiser interface centrally provides customisation operations which are delegated to the appropriate classes of the customisable package. In that way, the customiser interface builds on customisation hooks defined during micro customisation. Besides pre-defined customisation operations such as `enable()` and `disable()`, the designer is able to specify own customisation operations, e.g., `switchToTextMode()`. Since for plain UML packages the specification of interfaces is not allowed, we use in fact a special kind of package, namely *UML subsystems*

(denoted by the fork symbol, cf. Fig. 6). In general, we consider such customisation packages as part of the variable part of an application representing different views on the stable part which are adapted by means of customisation rules.

5.2 Notation for Specifying Customisation Rules

Customisation rules can be seen as a uniform mechanism for both, macro adaptation and micro adaptation. For this, a customisation rule is simply specified within an annotation, using the stereotype «`CustomisationRule`» (cf. Fig. 7). Each customisation rule consists, according to the generic rule model, of a name and an ECA-triplet.

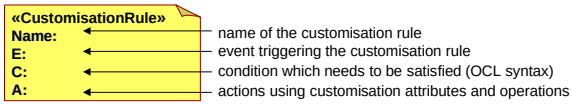

Fig. 7. Stereotyped Annotation for Specifying Customisation Rules

The annotation is attached to those model elements being subject to customisation, i.e., providing either the event for triggering the rule or the hook used by the action. According to our view of the customisation design process, customisation rules may be attached to model elements at content level, hyperbase level and presentation level [12]. The example depicted in Fig. 8 shows device-dependent customisation at the hyperbase level.

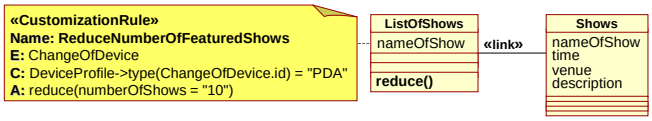

Fig. 8. Device-Dependent Customisation at the Hyperbase Level

Depending on the device used, the number of shows which are part of a single page "`ListOfShows`" is reduced to 10. For this, the event part of the rule incorporates the predefined event type `ChangeOfDevice`. Using profile information, the condition checks whether the device used was a PDA (note that additional rules would be needed to cover other device types).[1] The action uses the customisation operation `reduce()` to adapt the list of shows to the small device and consequently restricts the possible links to detailed information about shows.

5.3 Notation for Specifying Customisation Rule Patterns

To facilitate reusability of certain customisations, we introduce the notion of customisation rule patterns. Customisation rule patterns are some kind of parameterisable templates, predefining certain customisation rules in an application

[1] The syntactically precise name of the user agent profile used in the condition would be `CustomisationModel::ProfileModel::UserAgent`. For sake of readability we have used the shortcut `UserAgent` instead.

independent manner [13]. For their specification, the stereotype «CustomisationRulePattern» is used. Fig. 9 gives an example of a customisation rule pattern called ReducerPattern abstracting the device-dependent customisation rule of the example given in the previous section.

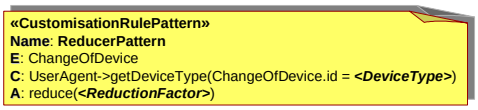

Fig. 9. ReducerPattern

The pattern reduces the number of list elements of a page according to a certain device type given. It takes two parameters namely the device type and the number of desired elements within the list.

6 Tool Support for Customisation

As any other modelling task within WUML, customisation design requires proper tool support. We propose an integrated modelling environment in terms of a *customisation toolkit* for the customisation design of ubiquitous web applications. Based on [20], this customisation toolkit should provide a *customisation rule designer* consisting of a set of graphical editors for defining and maintaining customisation rules, a *customisation rule browser* for facilitating the reuse of already existing customisation rules and an appropriate *repository* (cf. Fig. 10). Note that context modelling, profile modelling and the specification of customisation hooks should be supported by the tool constructed for designing the content level.

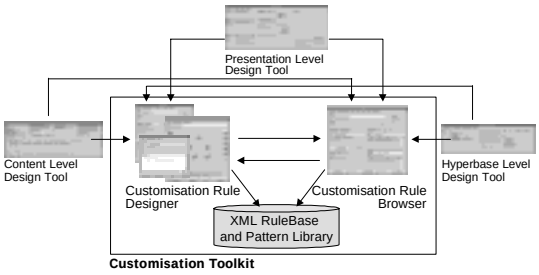

Fig. 10. Components of the Customisation Toolkit

It has to be emphasised that the components of the customisation toolkit should be *seamlessly integrated* among each other, as well as with the other tools provided by WUML. In particular, it should not only possible to navigate between the tools as depicted by the arrows in Fig. 10, but also to preserve the context during navigation. This implies that, e.g., a composite event selected in the customisation rule browser is automatically loaded when an editor of the customisation rule designer is opened. Concerning the relationship to the other WUML tools, it is envisioned that the customisation toolkit can be opened directly within each of them in order to retrieve and optionally modify existing rules in the context of certain model elements or to define new ones and attach them immediately to the appropriate model element using

the annotations described in Section 5. In the following, the envisioned components of the customisation toolkit are discussed in more detail.

Customisation Rule Designer. The customisation rule designer should allow for a graphical specification and modification of rules, together with their components. The customisation rule designer provides different editors, according to the components of a customisation rule (cf. Section 4.3). Specified rules are automatically translated into an appropriate XML specification [26], and stored within the *rule base* of the repository (cf. Fig. 10). Besides rules, the customisation rule designer should also support the definition and modification of customisation rule patterns and the semi-automatic generation of rules out of them.

Customisation Rule Browser. The customisation rule browser should support the designer of the ubiquitous web application in retrieving existing rules or components thereof. One of the major design goals of the browser is to provide an interface that can be used rather intuitively. In this respect, the browser should be able to answer the following queries, to mention just a few:

- Retrieve all rules *realising a requirement*, i.e., having a certain ECA combination
- Retrieve all *components of a certain rule*
- Retrieve all rules, that are *triggered by a certain event* (e.g. change of location)
- Retrieve all rules, that contain a *certain condition/action*
- Retrieve all *actions*, that might be executed as soon as a *certain event is signalled*
- Retrieve all rules realising customisation within, e.g., *presentation design*
- Retrieve all rules using the *customisation hooks* of a *certain model element*

Customisation Rule/Pattern Repository. The customisation rule/pattern repository and the envisioned process of working with them is shown in Fig. 11.

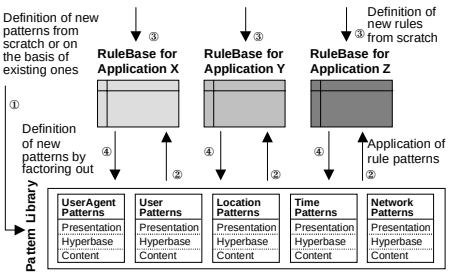

Fig. 11. Customisation Rule/Pattern Repository

In the long run, for each kind of customisation, appropriate rule patterns should be provided within a pattern library. The customisation rule bases and the customisation pattern library should be both organised as a set of dictionaries wherein all components are stored in terms of XML files. An important goal is to make the pattern library extensible allowing both, to define new patterns and to specialise existing ones (① in Fig. 11). At the same time, existing patterns or parts of them should be easily reused. In order to use rule patterns within an application, they have to be configured by the ubiquitous application designer. This is done by binding the parameters of a selected pattern on the basis of the application semantics. The system should guide the application designer during the process of parameter binding by providing on-line help for each required parameter and by restricting the binding

alternatives to those that do not contradict the specification of the underlying pattern. Once a rule has been fully and correctly specified, it can be automatically generated and stored in the rule base attached to the corresponding application (② in Fig. 11). Furthermore, it is still possible to specify rules without using a pattern and to store them directly within the appropriate rule base (③ in Fig. 11). This is normally done for rules without reusability in mind. If sometimes later an application designer recognises that a specific design situation recurs and thus is worth to be specified by a corresponding rule pattern, existing rules can be used for this abstraction process (④ in Fig. 11).

7 Outlook

In this paper we have introduced customisation as uniform mechanism for dealing with ubiquitous issues in web application development. Several issues remain to be resolved. We stress three of them. First, what is the "right" conceptual model of complex context and profile information? There are several standardisation efforts under way, such as CC/PP [24] and P3P [25], but to the best of our knowledge, there exists no comprehensive though open framework for context and profile modelling. In particular, we regard openness as crucial issue, since the kinds of relevant context and profile information will be growing in the near future (think of temperature in household systems, blood pressure in health systems, etc). Second, will there be an automatic recognition and computation of semantic equivalence? Which kind of modelling support is necessary? Resolving this issue is crucial for the success of "real" multi-delivery systems, where one can seamlessly switch from one delivery platform to another. Last but not least, where does customisation stop and redesign start? Since we are living in a changing world with changing requirements, continuous adaptation is almost a must. However, experience teaches us that continuously changing a software system will ultimately result in a complex inscrutable system. To find the right borderline between customisation and redesign is a challenging task.

References

[1] G. D. Abowd, Software Engineering Issues for Ubiquitous Computing, International Conference on Software Engineering (ICSE), Los Angeles, 1999.

[2] C. Avery, and R. Zeckhauser, Recommender Systems for Evaluating Computer Messages, Communications of the ACM (CACM), Vol. 40, No. 3, March 1997.

[3] C. Barry, M. Lang, A Survey of Multimedia and Web Development Techniques and Methodology Usage, IEEE Multimedia, Special Issue on Web Engineering, April, 2001.

[4] P. Brusilovsky, Adaptive Hypermedia: An Attempt to Analyse and Generalize, Multimedia, Hypermedia, and Virtual Reality: Models, Systems, and Applications, P. Brusilovsky, P. Kommers, and N. Streitz (eds.), Springer. Berlin, 1996.

[5] J. M. Carroll, and A. P. Aaronson, Learning by Doing With Simulated Intelligent Help, Communications of the ACM (CACM), Vol. 31, No. 9, September 1988.

[6] P. De Bra, Design Issues in Adaptive Web-Site Development, Proc. of the 2nd Workshop on Adaptive Systems and User Modeling on the WWW, Toronto, Canada, 1999.

[7] P. Fraternali, Tools and approaches for data-intensive Web applications: A survey, ACM Computing Surveys, Vol. 31, No. 3, September 1999.

[8] J. Gomez, C. Cachero, O. Pastor, Conceptual Modeling of Device-Independent Web Applications, IEEE Multimedia, Special Issue on Web Engineering, April-June 2001.

[9] M. D. Good, J. A. Whiteside, D. R. Wixon, S. J. Jones, Building a User-Derived Interface, Communications of the ACM (CACM), Vol. 27, No. 10, October 1984.

[10] G. Kappel, W. Retschitzegger, and B. Schröder, Enabling Technologies for Electronic Commerce, Proc. of the XV. IFIP World Computer Congress, Vienna/Austria and Budapest/Hungary, August/September 1998.

[11] G. Kappel, W. Retschitzegger, The TriGS Active Object-Oriented Database System - An Overview, ACM SIGMOD Record, Vol. 27, No. 3, September 1998.

[12] G. Kappel, W. Retschitzegger, W. Schwinger, Modeling Customizable Web Applications - A Requirement's Perspective, International Conference on Digital Libraries: Research and Practice (ICDL), Koyoto, Japan, November 2000.

[13] G. Kappel, S. Rausch-Schott, W. Retschitzegger, M. Sakkinen, Bottom-up design of active object-oriented databases, Communications of the ACM (CACM), 44(4), 2001.

[14] G. Kappel, W. Retschitzegger, W. Schwinger, A holistic view on web application development - the WUML approach, Tutorial notes at First Int. Workshop on Web-oriented Software Technology (IWWOST'01), Valencia, Spain, June 2001.

[15] G. Kappel, B. Pröll, W. Retschitzegger, W. Schwinger T. Hofer, Modeling Ubiquitous Web Applications – A Comparison of Approaches, Proc. Of the Int. Conf. on Information Integration and Web-based Applications and Services (iiWAS), Austria, Sept. 2001.

[16] B. Mobasher, R. Cooley, and J. Srivastava, Creating Adaptive Web Sites Through Usage-Based Clustering of URLs, Proc. of the 1999 IEEE Knowledge and Data Engineering Exchange Workshop (KDEX), November 1999.

[17] R. Oppermann, and M. Specht, A Nomadic Information System for Adaptive Exhibition Guidance, Proc. of the International Conference on Hypermedia and Interactivity in Museums (ICHIM), D. Bearman and J. Trant (eds.), Washington, September 1999.

[18] M. Perkowitz, and O. Etzioni: Towards Adaptive Web Sites: Conceptual Framework and Case Study, Proc. of the The Eighth Int. WWW Conference, Toronto, Canada, May 1999.

[19] B. Pröll, W. Retschitzegger, R. R. Wagner, and A. Ebner, Beyond Traditional Tourism Information Systems - TIScover, Journal of Information Technology and Tourism, Vol. 1, Inaugural Volume, 1998.

[20] W. Retschitzegger, TriGS Developer - A Development Environment for Active Object-Oriented Databases, 4th World Multiconference on Systemics, Cybernetics and Informatics (SCI), Orlando/USA, July 2000.

[21] W. Retschitzegger, and W. Schwinger, Towards Modelling of DataWeb Applications - A Requirements' Perspective, Proc. of the Americas Conferenc on Information Systems (AMCIS) Long Beach California, Vol. I, August 2000.

[22] J. Rumbaugh, I. Jacobson, G. Booch, The UML Ref. Manual, Addison-Wesley, 1998.

[23] M. Weiser, Some computer science issues in ubiquitous computing, CACM, 36 (7), 1993.

[24] World Wide Web Consortium (W3C), Composite Capabilities/Preference. Profiles, http://www.w3.org/Mobile, 2001.

[25] World Wide Web Consortium (W3C), Platform for Privacy Preferences (P3P) Project, http://www.w3.org/P3P, 2001.

[26] World Wide Web Consortium (W3C), eXtensible Markup Language (XML), http://www.w3.org/XML, 2001.