Ubiquitous Web Application Development - A Framework for Understanding

Anthony Finkelstein
Andrea Savigni
University College London
Department of Computer Science
a.finkelstein@cs.ucl.ac.uk
a.savigni@cs.ucl.ac.uk

Gerti Kappel
Werner Retschitzegger
University of Linz, Austria
Department of Information Systems (IFS)
gerti@ifs.uni-linz.ac.at
werner@ifs.uni-linz.ac.at

Wieland Schwinger
Software Competence Center Hagenberg (SCCH), Austria
wieland.schwinger@scch.at

Christian Feichtner
Siemens AG Österreich
Program and Systems Engineering
christian.feichtner@siemens.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 and personalisation calls for new engineering techniques supporting these kind of services. In this paper, we propose the notion of customisation as the uniform mechanism to provide the necessary flexibility with respect to both context-aware computing and personalisation. Customisation is realised in terms of a reflective architecture consisting of context, profile and customisation rule management.

1. Introduction

The Internet and the World Wide Web in particular have introduced a new era of computing, providing the basis for promising application areas like e-commerce [11], [19], and m-commerce [3]. At the beginning, the Web was merely employed for simple, read-only applications i.e., systems realised by a Web server offering static web pages for browsing only. Nowadays, the Web is more and more used as a platform for full-fledged, increasingly complex applications, where a huge amount of data is (partly) managed by underlying database systems and access to services can be ubiquitous i.e., from anywhere and any media at any time [1], [24].

What is a web application. For our purposes, a Web application is an application that was designed from the beginning to be executed in a Web-based environment. This apparently trivial definition reveals two very important aspects of such an application:

(1) A Web application is designed *from the start* to run in a Web-based environment. This means that the *hypermedia aspect* in terms of hypertext and multimedia in combination with traditional application logic must be taken into account throughout the application lifecycle, which makes it different with respect to a conventional application.

(2) A Web application is an application, not just a set of Web pages. In particular, this implies that it enforces the notion of a *session*, which differentiates it from the ordinary request-response Web paradigm. In this context, even a Web service that dynamically generates pages may not be considered a Web application. Think for example of a timetable service that, given desired departure and destination times and places, returns a set of pages containing the available trains and connections. In this case, there is no need for the service to maintain the notion of session which means that, in our view, this is *not* a Web application, but just a Web-based service.

What is an ubiquitous web application. In the context of the definition given above, an *ubiquitous web application* is a Web application that suffers from the *anytime/anywhere/anymedia syndrome*. This means that an 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 an ubiquitous web application must take into account the different capabilities of devices comprising display size, local storage size, method of input, network capacity, etc. 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, an ubiquitous web application must be, on

the one hand, *context-aware* i.e., aware of the environment it is running in, and on the other hand it must support *personalisation*.

Engineering ubiquitous web applications. Considering these kind of applications from a software engineering point of view, as their complexity increases, so does the importance of modelling techniques [8], [18], [21]. Building models of an ubiquitous web application prior to its construction is essential for fully understanding the application, for communication between 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.

Contribution. This paper is a suggestion as how this issue might be addressed. In particular, the contribution of the paper can be summarised as follows:

(1) We propose the notion of *customisation* as the uniform mechanism to provide adaptability (i.e., changing the application) with respect to both *context-aware computing* and *personalisation*.
(2) We provide a *framework* which is based on a *reflective architecture* since the use of reflection and metadata is essential for ensuring adaptability.
(3) We suggest a *holistic view on the development process of an ubiquitous web application* by introducing customisation as an additional modelling dimension influencing all other dimensions of web application development.
(4) We specially focus on requirements engineering by enforcing the distinction (borrowed from [5]) between goals and requirements. However, we consider requirements as *run-time entities*, that can change overtime under the influence of the environment.

Structure of the Paper. The paper is structured as follows. Section 2 discusses the previous work influencing our approach. Section 3 proposes a framework which is based on a reflective architecture. Section 4 introduces customisation as a new modelling dimension and gives insight into appropriate modelling concepts pointing the way to next-generation modelling methods for ubiquitous web applications. Finally, Sect. 5 draws some conclusions.

2. Background

The goal of this section is to give a brief overview of the main influences behind our work, namely Michael Jacksons's "world and machine" work [10], Axel van Lamsweerde's "Kaos" [5], computational reflection [15], web application engineering [21], and customisation approaches [13].

2.1. The World and the Machine

[10] represents a cornerstone in understanding the relationships between a software artefact and the surrounding world. Jackson identifies four facets of relationships between the world and the machine:

- the modelling facet, in which the machine simulates the world;
- the interface facet, where the world touches the machine physically;
- the engineering facet, where the machine controls the world;
- the problem facet, where the shape of the world and of the problem influences the shape of the machine and of the solution.

The discussion of the engineering facet turned out to be particularly useful to us, and particularly the distinction between requirements, specifications, and programs. Requirements are concerned solely with the world, programs are concerned solely with the machine, specifications are the bridge between the two. Section 3 will use these concepts in working out the boundaries between world and machine within our framework.

2.2. Goal-Oriented Requirements Engineering

The seminal works by Yue [27] and van Lamsweerde [5] opened a new direction in requirements engineering: the *goal-oriented* approach. The key achievement of this new approach is that it makes explicit the *why* of requirements. Quoting van Lamsweerde, "[before goal-oriented requirements engineering] the requirements on data and operations were just there; one could not capture *why* they were there and *whether* they were sufficient." [14] van Lamsweerde's approach provides three levels of modelling (meta level, base level, and instance level) that allow to represent the ultimate objectives of the system (the *goals*), and shorter-term, more concrete objectives (the *requirements*), that operationalise the goals.

2.3. Computational Reflection

"Computational reflection is the activity performed by a computational system when doing computation about its own computation." [15] A reflective system maintains, *at run-time*, data structures that materialise some aspects of the system itself, and that allow it to reason upon, and possibly change, itself.

For our purposes, reflection means that an explicit, run-time representation of system behaviour is maintained, which *reifies* the actual system behaviour in the sense that changes in the latter are materialised in the meta-level description. Similarly, changes in the meta-level description *reflect* back into the underlying system's behaviour. This "closed loop" approach is called *causal connection*.

Reflection, in our view, is a *mechanism*, not a goal. More precisely, it is a mechanism for manipulating meta data in a clean and consistent way. We regard reflection as key in this field because manipulating meta data is essential in this context of highly-dynamic services, as these must be able to dynamically adapt themselves to changing context and changing requirements.

2.4. Web Application Engineering

Engineering ubiquitous web applications is mainly addressed with modelling methods stemming from the area of developing traditional web applications that do not consider ubiquity. Basically, those methods can be considered along three orthogonal dimensions, comprising *levels*, *aspects* and *phases* [21].

Levels: content, hyperbase, and presentation. The first dimension of web application modelling comprises three different levels in terms of content level, hypertext level, and presentation level. The *content level* refers to domain-dependent data used by the web application and is often managed by means of a database system. The *hyperbase level* denotes the logical composition of web pages and the navigation structure. The *presentation level*, finally, is concerned with the presentation of the hyperbase level i.e., the layout of each page and user interaction. Note that the emphasis of each of these levels depends on the kind of web application being modelled.

Aspects: structure and behaviour. The second dimension comprises the aspects of *structure* and *behaviour*, which are orthogonal to the three levels of the first dimension. Concerning the content level, besides structuring the domain by means of standard abstraction mechanisms such as classification, aggregation and generalisation, the behavioural aspect in terms of domain-dependent application logic has to be considered, too. Similarly, at the hypertext level, structure in terms of page compositions and navigational relationships between them, as well as behaviour like computing the endpoint of a certain link at runtime have to be modelled. At the presentation level, finally, user interface elements and their hierarchical composition have to be modelled concerning the structural aspect. The behavioural aspect comprises modelling of reactions to input events, e.g., pressing a certain button as well as interaction and synchronisation between user interface elements. Note that similar to the levels discussed above, the amount of structure and behaviour which has to be modelled depends on the kind of web application.

Phases: analysis, design and implementation. The third dimension of modelling web applications comprises the different phases of a software life cycle, ranging from *analysis* via *design* to *implementation*. This dimension is orthogonal to the two previously presented ones, meaning that structure and behaviour of content, navigation and presentation has to be addressed in each phase of the development process. Currently, there is no consensus on a general process for web application development. However, the influence of technological aspects tailoring the model towards the implementation environment, such as distribution, heterogeneity and database aspects, should certainly increase within the later phases of the modelling process.

Shortcomings of Existing Methods. Along these dimensions a survey of eight approaches for web application engineering revealed the following shortcomings (for a detailed discussion see [21]):

- *Behavioural Modelling is Often Neglected*. Modelling the behavioural aspect of web applications at all levels is often neglected by existing methods. If behaviour is considered then mainly at the presentation level. Only those methods that are based on object-oriented modelling formalisms partly deal with behaviour modelling at all levels.
- *No Uniform Modelling Formalism*. With the exception of those few approaches which fully rely on the Unified Modeling Language (UML) [22], the majority of modelling methods is based on a mix of mainly proprietary modelling formalisms.
- *Presentation Level not Captured by Conceptual and Logical Modelling Concepts*. Most of the modelling methods do not support the presentation level with appropriate analysis and design concepts. Rather, authoring tools are often suggested for capturing the presentation level, thus loosing the benefit of technology independence.

- *No Process Support*. Most modelling methods do not follow a process for guiding the activities throughout the development of a web application.
- *Lack of Customisation Support*. Last but not least, one of the most severe drawbacks is the lack of concepts for customisation as needed by ubiquitous web applications. The various approaches focusing on customisation (cf. Sect. 2.5) are mainly implementation-oriented but do not provide proper concepts for the analysis phase and the design phases.

2.5. Approaches on Customisation

The following discussion is based on a broad view of customisation as discussed in [13] and tries to take into account various customisation issues encountered in application areas ranging from adaptive user interfaces [9] to adaptive hypermedia [2] and mobile computing. Especially the latter makes it necessary to consider not only the user preferences but also the environment in terms of, e.g., location in order to adapt the application [17].

We think that customisation could uniformly consider both *personalisation* and *context-aware computing*. Personalisation provides 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 if you want the added value) provided to the user should nevertheless be the same. In this case customisation enables to maintain *semantic equivalence*. This is why we talk, in this case, of semantic equivalence, which means that, despite the different context, the value provided to the user should still be the same.

3. The Framework

In this section, we propose a framework (shown in Figure 1) for developing ubiquitous web applications which is based on a reflective approach.

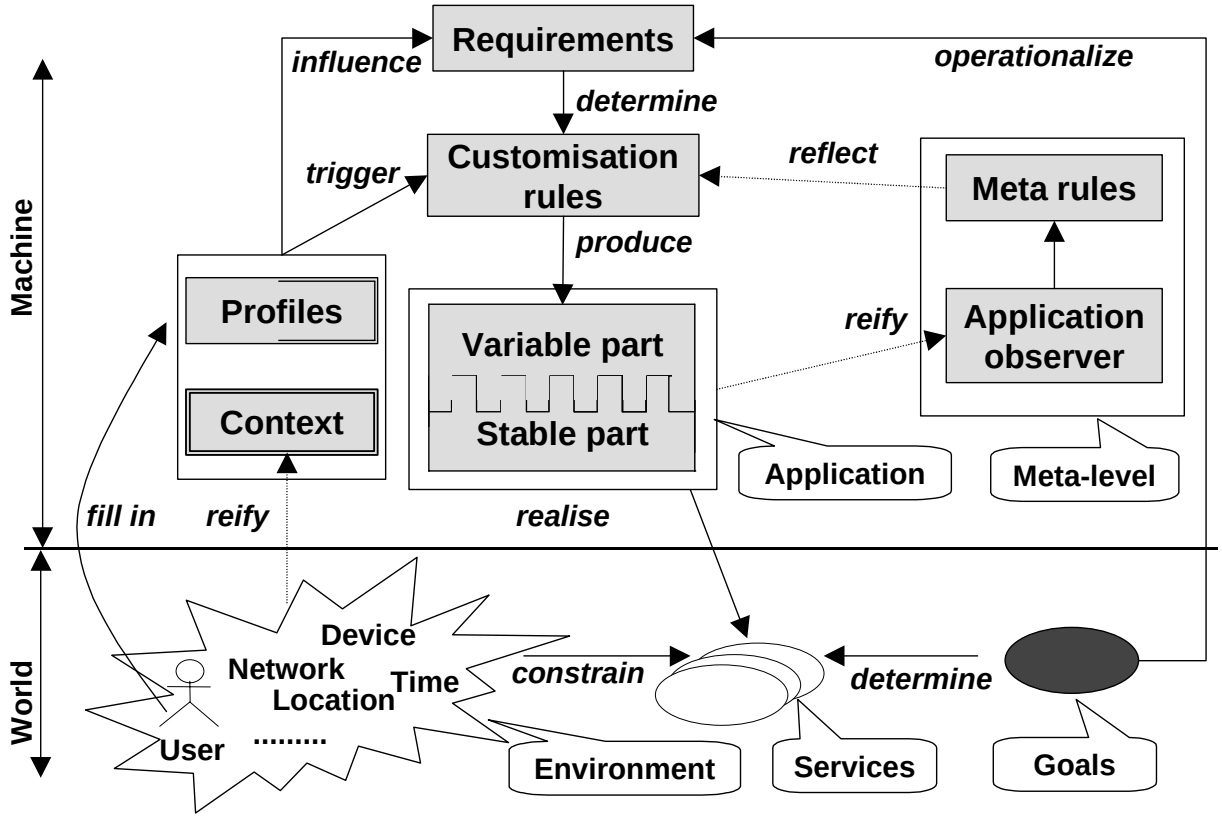

Figure 1. The Overall Framework

The rest of this section is devoted to a detailed explanation of the framework constituents. This explanation will follow a precise path that moves from the outside inward i.e., from the outer world towards the boundaries with the machine, and finally inside the machine itself.

3.1. Goal

A goal is an objective the system should achieve through cooperation of agents (user and software) in the software-to-be and in the environment. In our view goals are *immutable* i.e., they do not change with the changing environment. They represent the ultimate objective the service is meant to achieve. Changing the goals would mean changing the service itself. Along the lines of [5], a goal is not immediately achievable through actions performed by one or more agents; in other words, a goal is a somewhat abstract and long-term objective.

In Figure 1, goals are greyed out, which means that they are the only part of the framework that is not meant to change at runtime. More accurately, goals can (and should) have a runtime image (because their operationalisation into requirements may dynamically change) but they cannot be directly modified.

3.2. Service

The service can be defined as something that provides added value to one or more actors. A service is distinct from an application because it is a much more general concept, that could be provided e.g., by means other than computers. For example, if one of the goals is to maximise system usability, a service may be that of providing the user with specialised I/O devices. This is an example of user-centred design, whereby the system is just a means for meeting user goals, and not a goal on its own. Thus, services belong in the world, not in the machine.

3.3. Environment

By “environment” we mean whatever in the world provides a surrounding in which the machine is supposed to operate. Taking the environment into account is crucial because it strongly influences the behaviour of the machine. The environment comprises a number of *properties* each of which describes a different facet of the environment itself. Consider the example of an m-commerce service. In this case the environment comprises such things as bandwidth, location (absolute and relative), service availability, characteristics of the device, and many more issues.
Note that the system has no control whatsoever on the environment. In other words, the machine should adapt to the environment, not the other way around. As a matter of the fact, if the bandwidth is low, the connection is erratic, the PDA's display is small, this is something that cannot be directly changed by software. The job of a software engineer can therefore be summarised as a struggle *towards* the goal *despite* the environment; all we can do with the environment is sense it and describe it in the best possible way, but we cannot change it.

3.4. Context

Context is defined as the reification of the environment in terms of its properties. Note that in this case there is no reflection whatsoever (i.e., no downward arrow) because, as explained in the previous section, the environment is not modifiable. A context thus provides a manageable, easily manipulatable description of the environment. Most important, such description is continuously, dynamically updated to take into account the fact that the environment also continuously changes.

The context is part of the machine, as it is a representation of the environment (which itself belongs in the world) inside the system. In other words, the environment is represented in the context because there exists a machine; without this, the context would make no sense. Note that the context is the only part of the system that is *application independent*, in that it describes properties of the environment proper, regardless of the application that is to be built. This is marked in Figure 1 by a double border around the `Context` box. For a more detailed discussion see Sect. 4.1.

3.5. Profiles

Profiles, in contrast to the context, represent explicitly-given information. One prominent example of profiles for personalisation purposes are *user profiles*. A profile can comprise, on the one hand, information that is *voluntarily entered* by a user of the application or an administrator e.g., device characteristics; on the other hand, information that is *transparently acquired* by the system itself including e.g., usage statistics. Profiles can be either *application-dependent* or *application-independent*. For a more detailed discussion see Sect. 4.2.

3.6. Requirements

Requirements represent one of the possible ways of achieving a goal. A requirement operationalises a goal, in that it represents more concrete, short-term objectives that are directly achievable through actions performed by one or more agents. One key assumption that we make is that *requirements can change during system execution*, which further differentiates them from goals. In fact, due to a changing environment, the context may change in such a way that the operationalisation of the goals is no longer valid. This calls for monitoring of the context with respect to the goals: changes in the context may yield the necessity for changes in the requirements.

Both goals, on the one hand, and context and profiles, on the other hand, contribute to the operationalisation of goals into requirements. In very informal terms, one may say that requirements are a trade-off between the noble goals and the actual reality. For example, the goal of an ubiquitous m-commerce application might be to provide for a highly interactive user experience. Given this goal, if the context is favourable (e.g., high bandwidth, large colour display, Java Virtual Machine available) a requirement might be “use a colourful Java applet to represent the state of the shopping basket,” whereas if the connection is slow or there is no JVM available, the requirement may be mitigated into “use a 16-colour animated gif.”

3.7. Application

The application is the heart of the machine. It implements one or more services. Note that the relationship between application and service is a many-to-many one, as one application could provide many different services; on the other hand, a complex service may need several distinct (and potentially distributed) applications to be realised.

An application is made up by a *stable part*, that provides the basic functionality, and a *variable part*, that realises the customisation. The stable part of the application provides hooks, so called *customisation hot-spots*, that allow the variable part to adapt its behaviour to the particular requirements. Under this respect, it resembles an object-oriented framework.

3.8. Customisation Rules

Customisation rules are the means by which requirements are translated into the variable part of the application and can be thought of as production rules (see Sect. 4.3).

3.9. The Meta Level

The meta level is a representation of the application in reflective terms. It implements the causal connection with respect to the application, in that it *reifies* the state of the application or, in other words, it continuously monitors the application and maintains an up-to-date description of its state. This is necessary because the application functioning can be influenced by the environment, and can therefore change in unpredictable ways. Suppose for example that a mobile phone is operating in an area with strong magnetic fields. In this situation, the phone may report a good connection but the actual quality provided may be low due to interference. In such a case, monitoring the environment is not enough, and there is no way to notice this interference other than monitoring the application itself.

The relationship with the application is two-way, in that the meta level can actually affect the base level functionality by means of *meta rules* i.e., rules whose domain is the base-level customisation rules. When faced with particular situations, the meta rules are triggered and can change or remove existing rules, or add new ones. A reflective meta level can also be employed for requirements monitoring, in the sense used in [7].

The meta level is not the primary focus of this paper and will therefore not be further discussed.

We argue that the “reflective way” is a clean and consistent manner of performing run-time changes to the underlying level (which is finally the actual system as perceived by the user). This approach consists in manipulating the customisation rules in order to reconcile the service with the requirements. The causal connection, in particular the downward link (reflection) provides for the consistency between the service description and the service itself. Architectural reflective techniques can be employed to that aim [23].

4. Customisation as a New Modelling Dimension

As already shown in the framework, customisation allows the incorporation of requirements into ubiquitous web applications. To provide a *holistic view on the development process of an ubiquitous web application* we introduce customisation as an additional modelling dimension influencing all other dimensions of web application development (see Figure 2). This means that customisation has to be considered for all aspects, levels and phases of a web application.

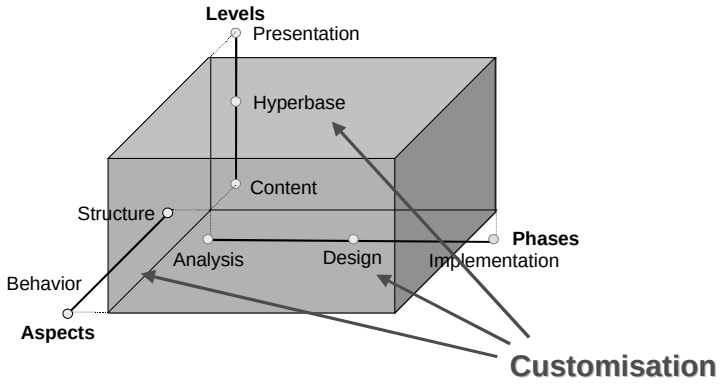

Figure 2. Scope of Customisation

Taking into account the framework presented in Sect. 3 in terms of modelling, three main aspects of customisation have be captured by proper modelling concepts, namely *context*, *profile* and *customisation rules*, which are discussed in more detail in the following.

4.1. Context Modelling

As previously presented, existing approaches for customisation suggest to capture various context properties [13]. Based on these existing literature, we propose that context modelling should at least consider the following *context properties*:

- *User Agent:* The user agent property refers to the demand for multi-delivery. It provides information about the device and browser. Together with a device profile and a browser profile (see Sect. 4.2) this allows to identify constrains relevant for multi-delivery.
- *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 (see Sect. 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 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 should 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 context modelling 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 as done in [16] 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 Figure 3 by means of a UML class diagram.

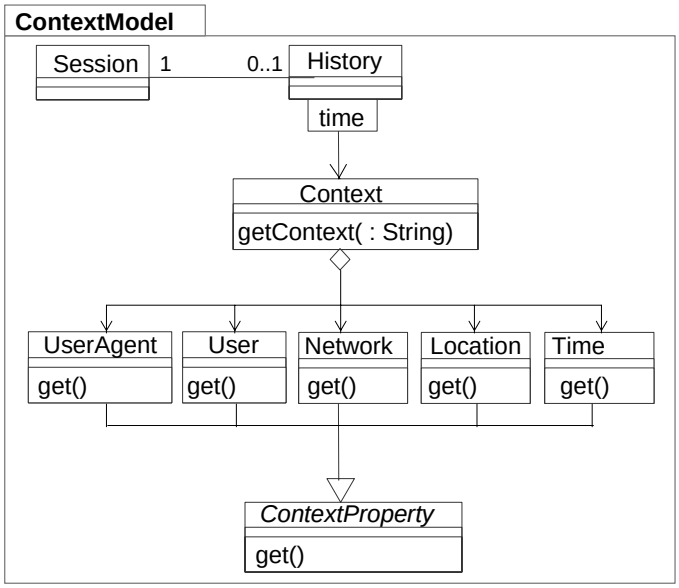

Figure 3. The Context Model

The context, represented by the `Context` class, 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. We chose this apparently naive representation because we know that we won't be adding new subclasses in the near future and we prefer to have a *context vector* that explicitly represents the specific subclasses. Note that, should the necessity arise to add new subclasses, the change required to the `Context` class would be very limited compared to that required on the customisation rules side.

4.2. Profile Modelling

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)* [25] which is based on the Resource Description Framework (RDF) [26]. 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 to be extensible 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. In the domain of ubiquitous web application, at a minimum a user profile and a device profile should be considered. Since the profile information might be application-dependent the software engineer 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 content information.

Analogous to the context model described in the previous section, our profile model encompass the following profiles (see Figure 4):

- *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 comprises information about the user's preferences with respect to personalization. Note that in general the user profile is application dependent as opposed to the user agent profile and the context.

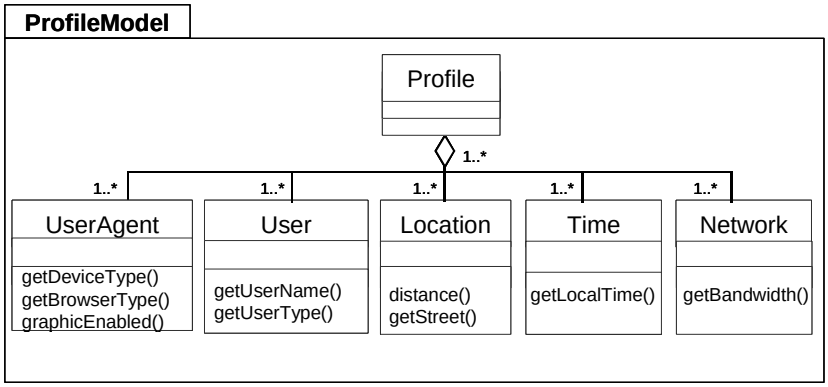

Figure 4. The Profile Model

- *Network:* A network profile is application-independent and could contain e.g., 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.

4.3. Customisation Rules

As already mentioned, customisation rules are the means to translate requirements into the variable part of the application. For specifying customisation rules, we propose similar to [4] the usage of the event/condition/action (ECA) mechanism which is well known in the area of active database systems [12] (see Figure 5). The event part of the rule determine the events which are able to trigger the rule. If the rule incorporates a condition, the condition is evaluated as soon as the rule is triggered. If the condition evaluates to true, the rule's action is executed.

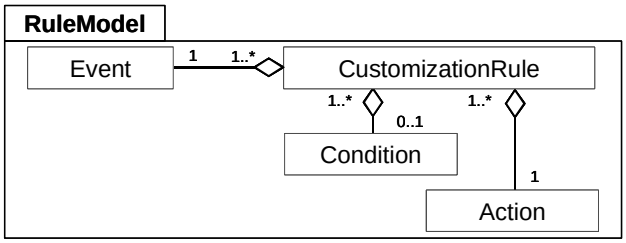

Figure 5. . The Customisation Rule Model

Event Model. Applying the ECA mechanism in the domain of ubiquitous web application requires a dedicated event model. The events of this event model need to monitor changes within both, context and profile information as well as explicit user requests. Consequently, our event model considers three basic types of *primitive events* (see Figure 6):

- *Changes in the 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.
- *Changes in the profile*: Since the properties of certain profiles are not necessarily limited, we propose at a first attempt one pre-defined event only, namely `ChangeOfProfileProperty`.
- *Request of a User*: Each time the user interacts with the system (e.g., activating a link to another page), a request to ubiquitous web application is generated. Such a request signals the pre-defined event `userRequest`.

To be able to model complex real-world situations we suggest the notion of so-called *composite events*. Composite events are constructed by means of the logical event operators `AND`, `OR`, and `SEQ`, in addition to the above mentioned primitive events. For example, for a page which should be both personalised and optimised for different devices a

composite event like `(ChangeOfUser OR ChangeOfDevice)` would be required. Composite events also allow expressing 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 the context and changes in the profile only, whereas in the latter, the request of a user event has to be taken into account too. Several properties of a rule, such as *priority*, *activation state* and *transaction mode*, 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.

Figure 6 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 model by means of subclassing.

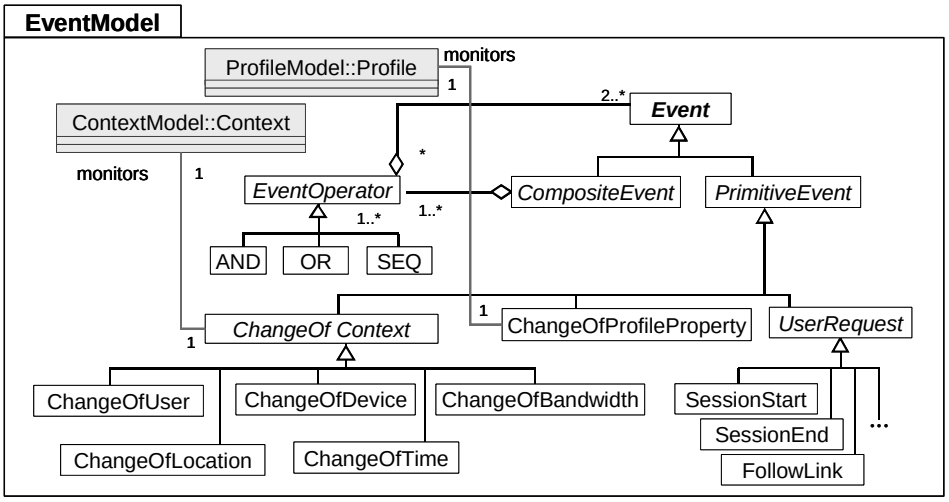

Figure 6. The Event Model

Condition. Whereas events specify when adaptation is potentially necessary due to changes in context or user profile or due to a user's request, conditions determine whether (and which) adaptation is actually desired by considering both context and profile information. Thus, conditions are the means to test whether a certain requirement is violated or not. Conditions are in fact predicates which can be combined by means of logical operators.

Action. Actions are used to realise necessary variations i.e., they represent a sequence of activations of the hook methods provided by the stable part of the application. Thus, actions generate the variable part of the application, e.g., an additional link at a certain page. In this way, actions ensure the fulfilment of requirements.

Examples**.**

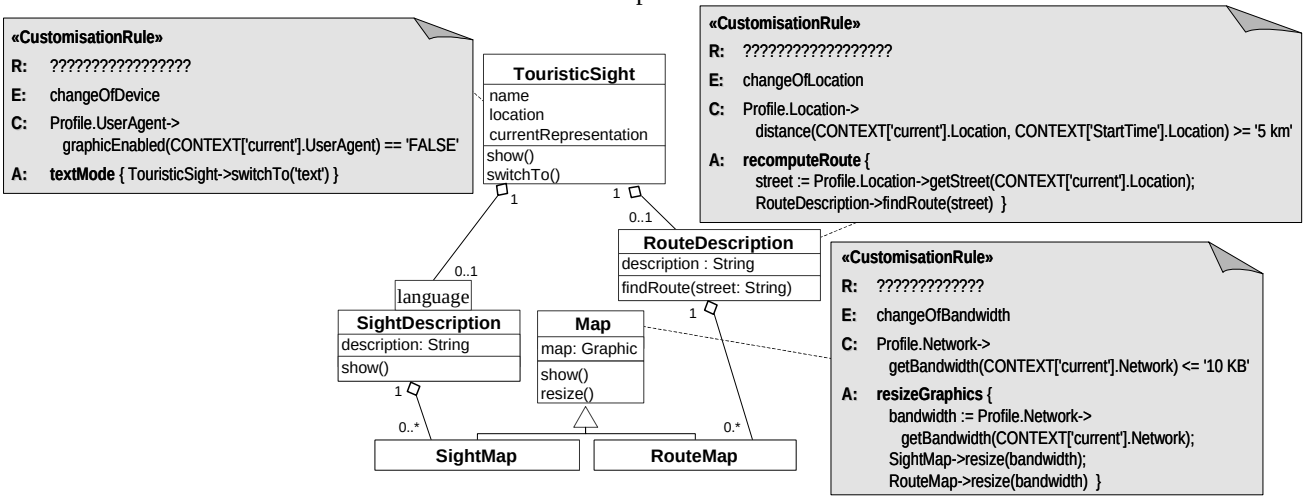

Figure 7 shows example model with some customisation rules. The classes are modelled using UML. The customisation rules are formulated in pseudo code and are attached to the modelled classes by annotations. Those annotations are stereotyped with «CustomisationRule» to indicate their function. The specification of the requirement the customisation rules is realising is marked with "R:". The "E:", "C:", and "A:" mark the event, condition, and action of the customisation rule, respectively. The first rule specifies the requirement to use text only on non-graphic enabled devices. The event detects that the device changed, the condition evaluates the graphical capability of the device by accessing the device's profile and the action activates the hook method switchTo() of the customisable object TouristicSight.

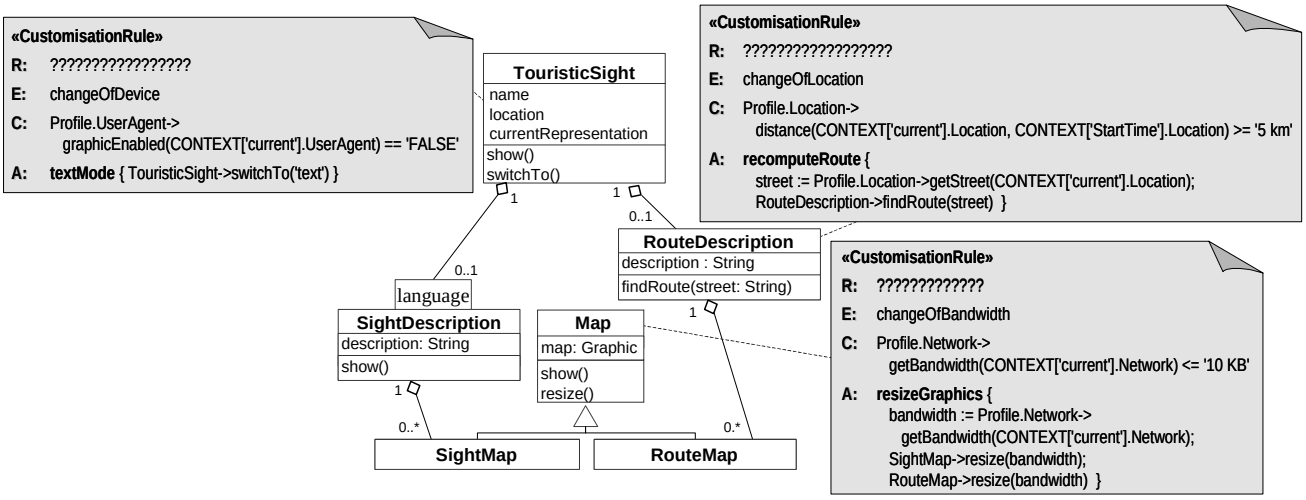

Figure 7. Examples of Customisation Rules[1]

The second rule customises the graphic resolution according to the bandwidth. For this, the event detects bandwidth variations, the condition checks whether the bandwidth falls below 10 KB, and the action resizes the two maps (SightMap and RouteMap) proportionally. The last rule recalculates a route if the user moves 5 km away from those position the calculation of the route was based on. The rule is triggered by the user's change of the location, the condition uses the method distance() to calculate the distance between the current position of the user and the position the route was calculated before. The action re-computes the new route taking the user's current position as input parameter.

5. Conclusions

Ubiquitous web applications are web applications suffering from the anytime/anywhere/anymedia syndrome. This means that engineering such ubiquitous web applications need to take into account specific constraints arising from the application's environment and the preferences of the individual users. Customisation can serve as a uniform mechanism to tackle these constraints.

This paper has proposed a framework for understanding for developing ubiquitous web applications based on a reflective approach. Such applications are modelled as machines which interact with the world and which comprise components that allow reflection on the world. Our approach also considers customisability. On the basis of this framework a holistic view on the development process of ubiquitous web applications has been presented incorporating customisation as additional development dimension next to the three orthogonal dimensions of levels, aspects, and phases. Customisation will consist of context, profile and customisation rule management. First ideas on handling these three components during the development process have been presented.

Acknowledgements

This work was partially funded by *UWA (Ubiquitous Web Applications)*, an EU-funded Fifth Framework Programme project (IST-2000-25131) that the authors are carrying on in co-operation with a number of academic and industrial

[1] For readability reasons, we have introduced CONTEXT[<time>] as a shortcut for session.history[<time>].context.

partners from six European countries. The work described in this paper is intended as an initial contribution to the project.

References

[1] G. D. Abowd, E. D. Mynatt, "Charting Past, Present, and Future Research in Ubiquitous Computing", *ACM Transactions on Computer-Human Interaction*, Vol. 7, No. 1, March 2000.

[2] 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.

[3] D. Chakraborty, H. Chen, "Service Discovery in the future for Mobile Commerce", *ACM Crossroads*, Winter 2000.

[4] S. Ceri, P. Fraternali, and A. Bongio, "Web Modeling Language (WebML): a modeling language for designing Web sites", *Proc. of the 9th World Wide Web Conference (WWW9)*, Amsterdam, May 2000.

[5] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed Requirements Acquisition", *Science of Computer Programming*; Vol. 20, 1993.

[6] G. Ehmayer, G. Kappel, and S. Reich, "Connecting Databases to the Web - A Taxonomy of Gateways", *Proc. of the 8th Int. Conf. on Database and Expert Systems Applications (DEXA)*, France, Springer LNCS 1308, September 1997.

[7] S. Fickas, and M.S. Feather, "Requirements Monitoring in Dynamic Environments", *Proc. of the Second IEEE International Symposium on Requirements Engineering*, 1995.

[8] A. Ginige, D. B. Lowe, and J. Robertson, "Hypermedia Authoring", *IEEE Multimedia*, Vol. 2, No. 4, 1995.

[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] M. Jackson, "The World and the Machine", *Proc. of the 17th International Conference on Software Engineering*, Seattle, Washington, USA, April 1995.

[11] 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.

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

[13] 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.

[14] A. van Lamsweerde, "Requirements Engineering in the Year 00: A Research Perspective", *Proc. of th 22nd International Conference on Software Engineering (ICSE'2000)*, 2000.

[15] P. Maes, "Concepts and Experiments in Computational Reflection", *Proc. of OOPSLA87*, *Sigplan Notices, ACM*, Oct. 1987.

[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] T. Powell, *Web Site Engineering*, Prentice Hall, 1998.

[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] Rational Software Corporation, The Rational Unified Process, http://www.rational.com/products/rup/.

[21] W. Retschitzegger, and W. Schwinger, "Towards Modeling 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, and G. Booch, *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1998.

[23] F. Tisato, A. Savigni, W. Cazzola, and A. Sosio, " Architectural Reflection. Realising Software Architectures via Reflective Activities ", *Proc. of the 2nd Engineering Distributed Objects Workshop (EDO 2000)*, California, USA, November 2000.

[24] M. Weiser, "Some computer science issues in ubiquitous computing", *CACM*, Vol. 36, No. 7, July 1993.

[25] World Wide Web Consortium (W3C), Composite Capabilities/Preference. Profiles, http://www.w3.org/Mobile, 2000.

[26] World Wide Web Consortium (W3C), Resource Description Framework (RDF), http://www.w3.org/RDF, 2000.

[27] K. Yue, "What Does It Mean to Say that a Specification is Complete?", *Proc. of the Fourth International Workshop on Software Specification and Design (IWSSD-4)*, Monterey, CA, USA, 1987.