# Ubiquitous Web Applications

UWA Consortium[1]
*Piazza del Carmine 22, 09100 Cagliari, Italy*
*Tel: +39 0574 27256; Fax: +39 0574 401443; Email: gab@acm.org*

**Abstract.** Web applications have already evolved from *static* sites to completely distributed applications; nowadays they are facing a new transformation and are becoming ubiquitous systems that are available anytime, anywhere, and with any media.
This new requirement led the *Ubiquitous Web Applications* (UWA) project (IST-2000-25131) to propose a special purpose design approach to modelling Web applications.

## 1. Introduction

We are facing the need for new web applications enabling ubiquitous access to e-commence and m-commerce services [1]. Ubiquitous computing was first stressed by Marc Weiser [2], envisioning a scenario where computational power would be available everywhere embedded in walls, chairs, clothing etc. Weiser's goal is to achieve the most effective kind of technology, which is available throughout the physical environment, while making them effectively invisible to the user. In the area of web applications, ubiquity is not seen as visionary in this highly pervasive sense, meaning that computing power is embedded everywhere. Rather, ubiquitous web applications build more on existing technology, in that web access is no longer primarily a domain of browsers based on desktop PCs but more and more done by various commercially available mobile devices. In general, ubiquity offers new opportunities and challenges for web applications in terms of *time-aware* [3], *location-aware* [4], *device-aware* [5] and *personalised services* [6]. This implies that ubiquitous web applications have to take into account, individually for each user, time and location of access, together with the different capabilities of devices comprising display resolution, local storage size, method of input and computing speed as well as network capacity. Consequently, the fundamental objective of ubiquitous web applications is to provide services not only to people at any time, anywhere, with any media but specifically to communicate the right thing at the right time in the right way.

Given this belief, the UWA consortium started working on the special-purpose modelling methodology that is presented in this paper.

The overall modelling problem is partitioned into the following design aspects:

- *Requirements elicitation* to define what the application should do;
- *Hypermedia Design* to model data, and how they can be navigated and presented, and operations (services) as available to the user;
- *Transaction Design* to model the transactions made available by the application;
- *Customisation Design* to specify how the application should adapt itself to the context, and, in particular to the user, device, communication channels, time and location.

Each modelling activity is defined in terms of a *metamodel*, which captures the set of relevant concepts and the primitives; *a notation*, based on UML [7] to represent the concepts; *a set of guidelines and heuristics*, to help the designer exploit the concepts and understand the trade-off among the different design solutions; and a *set of tools*, to enact the design process and enforce coherence and consistency of design.

The UWA project provides a *unified framework*, which integrates the various metamodels and notations and highlights their mutual interdependence, and *a unified software* environment, based on Rational Rose, that integrates the tools specific to each modelling activity. These tools have been developed by extending the Rational Rose CASE tool through its REI mechanism (Rose Extensibility Interface).

## 2. Requirements Elicitation

[1] The UWA consortium (www.uwaproject.org) comprises: Atlantis SpA (Italy), Banca 121 (Italy), Fundacion Robotiker (Spain), Politecnico di Milano (Italy), Punto Comercial (Spain), University of Linz (Austria), Technical University of Crete (Greece), Siemens AG (Austria), Università della Svizzera Italiana (Switzerland), and University College London (United Kingodm).

The major influence on our approach to requirements engineering is Axel van Lamsweerde's KAOS work [8]. Work by Lamsweerde (and Yue [9]) introduced a new approach to requirements engineering. Their *goal-oriented* approach makes the *why* of requirements explicit by tying requirements to goals. A **goal** is a somewhat abstract and long-term objective the system should achieve through cooperation of agents (user and software) in the software-to-be and in the environment, while **requirements** are shorter-term and more concrete objectives. Requirements *operationalise* goals.

Goals and requirements are to be placed within a framework which conceptually supports the elicitation of goals and the refinement of goals into requirements. Key aspects of this framework are introduced in the following.

A **stakeholder** is someone or something that has an interest in the system. This definition is purposely very vague because a stakeholder is an extremely general concept. Almost anyone can be a stakeholder. Examples include end users, developers, buyers, managers (i.e., people who will not use the system but will manage people who do). These stakeholders are very important but too often neglected in the requirements engineering process. A stakeholder owns one or more goals, and a goal may be owned by one or more stakeholder. A goal that interests no one is a non-goal, and should therefore be removed. Given the nature of the applications involved, a user-centred approach is employed. It means that the centre of our world is no longer the system, but rather the stakeholders of the system.

Next, a goal delivers a certain **value** to its stakeholders. The actual value delivered represents the *why* of the ownership, and is strictly dependent on both the goal and the stakeholder. The value is extremely hard to formalise. Therefore, it is usually expressed in prose as a comment. It is an arbitrary quantity that cannot be taken as an absolute measure. It is nonetheless very useful for establishing importance and priority of goals.

High-level goals represent the ultimate desires of stakeholders. However, for them to be of use, they have to be **refined** into lower-level goals. This refinement process is extremely useful because a high-level goal *per se* does not say much to the designer. It is too abstract, too high-level and too long-term to be fed directly to Web designers. In addition, refining the goals into subgoals is invaluable for eliciting new requirements, and assessing existing ones. In our experience, it is often very hard to understand what the real goal of the customer is.

Refining a goal into subgoals helps identify **conflicts**. A conflict is a relationship between two goals or requirements that means the two cannot be fulfilled together. A conflict must be solved as soon as possible, and in any case before the operationalisation step, that is, before any of the goals involved in the conflict are turned into actual requirements.

Requirements – the leaves of the derivation graph – are categorised into **dimensions**. As already stated above, this is the first case in the literature in which the goal-oriented approach has been applied to interactive systems and Web-based applications, and thus a novel requirement categorisation scheme had to be invented.

Each requirement belongs to exactly one dimension. This restriction can also be seen as a necessary (although certainly not sufficient) condition for a requirement to be considered as such: if a requirement cannot be easily and clearly assigned to exactly one dimension, then it is too general to be called a requirement (and is therefore still a goal).

Requirements also have an associated **priority**. Prioritising requirements becomes very desirable in any realistic software engineering methodology.

Finally, an **assumption** represents some entity, event, or other piece of information that belongs to the world and that we have to come to terms with when refining goals into subgoals and eventually into requirements.

## 3. Hypermedia Design

Hypermedia aspects in UWA are dealt with by suitably tailoring W2000 [10], whose main concepts are organised in three main models.

### *3.1 Information Model*

The **Information model** specifies the concepts for specifying the content available to the user (Hyperbase) and how it can be accessed (Access structures). The key element is the **Entity**. It renders data of interest to the user as if they were conceptual objects. An entity resembles the concept of a class and, as classes, it can be the root of a generalisation hierarchy. An entity is organised in semantic sub-units, called **Components**, which are pure organisational devices for grouping the contents of an entity into meaningful chunks. The result of this definition is a tree of components, based on the part-of relationship. Components can further be decomposed in sub-components, but the actual contents can be associated with leaf components only. The contents of leaf components is defined in terms of **Slots**, i.e., the attributes that define the primitive information elements. A **Segment** groups slots to supply information chunks as *consumed* by the user.

A **Semantic Association** connects two entities with a double meaning: it both creates the *infrastructure* for a possible navigation path (by connecting a source to a target) and has proper, local, information, called **Association Center**, which contains data that define and specify the association itself and provides additional information on how to represent both the single target elements, in a concise way, and the whole group of target elements that relate to the same source. Entities can also be grouped in **Collections** that are organised sets of information objects. A collection provides the user with a way to explore the information contents of the application and, thus, is the key concept as to access structures.

### *3.2 Navigation Model*

The **Navigation model** specifies the concepts that allow the designer to reorganise the information for navigational purposes. He should *reuse* the elements in the previous model to specify the actual information chunks together with the relationships among them. The information content is organised in atomic units, called **Nodes**. They do not define new contents, but either come from entity components, semantic association, and collection centers, or are added only for navigation purposes (e.g., to introduce fine-grained navigation steps). In the former case, they contain the slots associated with the information element they render. In the latter case, they are simple empty nodes. Two nodes are linked through a directed **Accessibility Relationship** to specify that the user can navigate from the source to the target node.

Nodes exist in the context of a **Navigation Cluster** that groups nodes and accessibility relationships to foster and facilitate the navigation among data (nodes). Clusters can be nested and can further be characterised according to the kind of information they render. **Structural Clusters** consist of all the nodes derived from the components of entities, **Semantic Clusters** comprise all the nodes that come from source, target and centers of semantic associations, and **Collection Clusters** comprise all the nodes that come from the members and centers of collections.

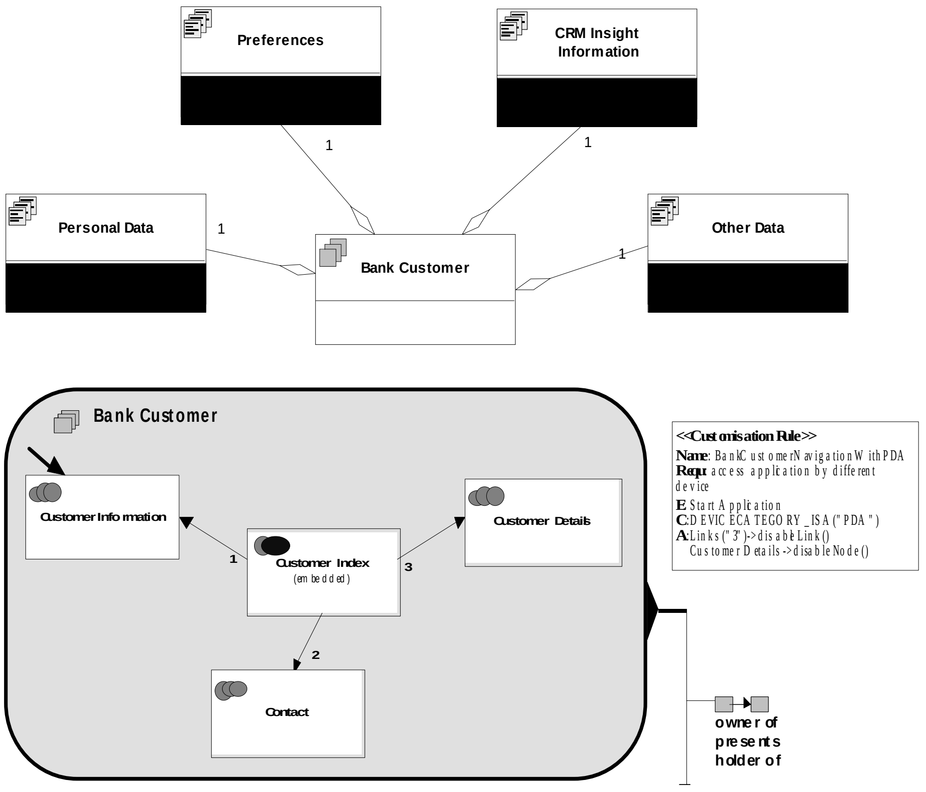

**Figure 1:** Information Design and Structural Navigation Design examples

### *3.3 Presentation Model*

The **Presentation Model** defines the concepts needed for the designer to specify how the content is published in pages and how users are supposed to reach data within the same page or across different pages.

**Presentation Units** are the smallest units at presentation level. They can either come from nodes or add new content that is defined only at presentation level for aesthetic/communication purposes. A **Section** is a set of presentation units derived from nodes that belong to the same navigation cluster. A **Page** is a grouping of sections, which could also be non-semantically related, from which it inherits links and navigation features. Presentation units, sections, and pages can all be sources or targets of **Presentation Links**, that is, a connection between two presentation elements to enable the navigation between them. According to the aforementioned concepts, we can further classify the links in a page as **Focus Links** to remain in the same page, but moving the page focus from one unit to another, **Intra-page Links** to navigate between instances of the same page type, and **Page Links** to navigate between instances of different page types.

One of the main differences of Web applications, with respect to more traditional Web sites, is the possibility of invoking special-purpose operations (services) while browsing the site. Operations can change the hypermedia and business states of the application, but they can also affect the underlying system, control or be controlled by external elements (e.g., an S.M.S. server), and be either explicitly triggered by users or implicitly invoked in particular situations. In UWA, designers can add:

- *Simple Operations*, which are atomic and must be considered a black-box component with respect to the user's point of view.
- *Multi-step Operations*, which preserve their essence of being atomic, but are not black-box anymore.
- *Activities*, which are not atomic anymore. They can be seen as business transactions and/or containers for operations (both simple and multi-step ones).

In Figure 1, an example of a piece of design is showed. For lack of space, we show only the in-the-large diagrams (without details) and leave out the relative presentation diagrams. However, in the picture it is possible to see the relation between the Information Design and the Navigation Design. Moreover, the navigation diagram contains also a customisation rule with the aim of showing an example of the relations with the Customisation Design.

## 4. Transaction Design

One of the most important success factors for e-business is the transactional behaviour that each Web application offers. Frequently such behaviour is complex, composed of several sub-transactions and accesses many different and distributed resources including existing legacy systems. Transaction design for such applications needs to be very flexible allowing both the development of Web applications from scratch, by decomposing user goals into sub-goals that exhibit transactional behaviour (top-down design), and using already existing systems or services to compose new applications offering added value services (bottom-up design).

**Extended Transaction Models** (ETMs) provide for transactions with complex internal structures and up to now several different such models have been proposed (sagas, nested, open nested, etc). Also, some recent web standards have been adopted and new proposals are continuously appearing. However, they have limitations that make their use for advanced Web applications difficult. Limitations come mainly from their inflexibility to incorporate different transactional semantics in one (structured) transaction or to describe different behavioural patterns for different parts of the same transaction. On the other hand, there is no high-level design mechanism to facilitate the transaction design process at an application design level.

Our objective is to facilitate the complex design process for web transactions by providing a **high level transaction modelling language based on UML extensions** for designing complex web transactions. To achieve this we propose UTML (Unified Transaction Modelling Language) as a high level transaction modelling language that can be used by an application designer to describe transactional behaviour according to the application's requirements and complexity. UTML is based on a transaction metamodel, which is flexible enough to describe transactions conforming to most of the known transaction models. This metamodel provides the appropriate mechanisms and primitives that are needed for describing complex transactional behaviour and is provided to the designer through a rich UTML notation (a set of UML extensions for the purpose of transaction modelling). UTML describes transactions from an application point of view and this has many advantages since the transaction itself is considered to be involved in the whole application and designed in this way. In particular UTML:

- Provides description for both structural and execution dependencies of transactions. That is, it can model what activities a transaction includes and with what semantics, when it can be invoked, who can invoke it, etc.
- Provides detailed specification of transaction decomposition semantics not for the whole model necessarily, but for each transaction node independently. This is important since it allows for incorporating behaviour of different transaction models into the same structured transaction.
- Distinguishes between management operations and functional operations that a transaction has, giving the ability to specify its behaviour.
- Provides for designing transactions with execution contracts weaker than ACID, integrating diverse resources like legacy systems. Moreover, it formalises the decomposition of such transactions and the propagation of ACID properties in sub-transactions.
- Introduces the concept of well-formedness rules that are applied on well-described concepts and are used to describe intra- and inter-transaction dependencies.
- Is an extensible modelling language for describing application-specific transactional behaviours. Well-formedness rules and management operations compose the extensibility mechanism of UTML.
- Describes transaction execution flows and run time execution dependencies between transactions using finite state machines.
- Provides a rich UML based notation, with appropriate stereotypes, that is used to visualise and document the transaction design.

## 5. Customisation Design

Our approach to *customisation design* is based on a broad view on customisation [11]. Although most often separated in existing approaches [12], we think that customisation for ubiquitous web applications should uniformly consider *personalisation* aspects, together with issues resulting from being *ubiquitous*, thus supporting the *anytime/anywhere/anymedia paradigm*. In our opinion, the design space of customisation comprises the two orthogonal dimensions **context** and **adaptation**. The context dimension comprises the circumstances of consumption of a ubiquitous web application mainly dealing with the question “why to customise and when”. In this respect, we define context as the *reification of certain properties*, describing *the environment of the application* and *some aspects of the application itself*, which are necessary to determine the need for customisation. The adaptation dimension mainly refers to questions concerning which changes to make as well as what to change. Customisation is seen, in turn, as a combination of a certain context and certain adaptation, thus adapting the ubiquitous web application towards a certain context. To accomplish this we base our approach on a reflective architecture as depicted in Figure 2. The *context* provides detailed information about the environment of a web application and the web application itself. Thereby the context influences not only the requirements as gathered by requirements elicitation but also triggers the actual customisation as soon as the context changes. The context is divided into a **physical context** representing the level of environment sensors and the **logical context** representing abstracted information. A rule-based mechanism in terms of **customisation rules** is employed in order to specify the actual customisations. Customisation rules are, in turn, determined by requirements. For providing a separation of concerns between the primary service requirements and the requirements of ubiquity, the application is divided 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.

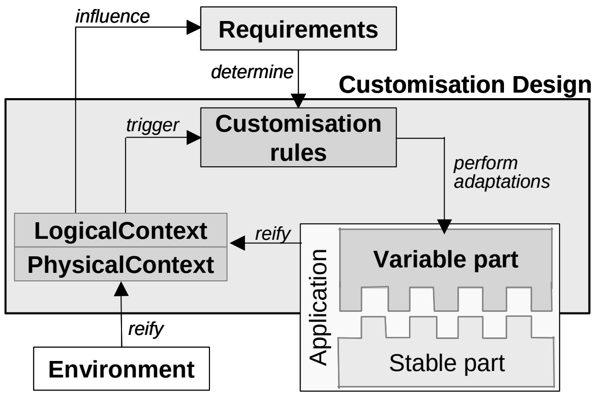

**Figure 2:** Overall Architecture of Customisation Design

To support the architecture introduced in Figure 2, we propose a **generic customisation model** in the sense of an object-oriented framework, which provides some pre-defined classes and language constructs in order to model application dependent customisation. For example, the *customisation rule model* allows specifying certain customisations. The adaptation desired towards a certain context is specified in terms of *customisation rules* which are specified within UML annotations attached to those model elements being subject to customisation. The customisation rule model again provides a set of sub models in terms of an **event model**, a **condition model** and an **action model**. The event model specifies a set of pre-defined events, responsible for determining potential violations of certain requirements due to changes in context. The condition model provides logical expressions using OCL syntax and allows specifying predicates on the context model. The action model, finally, defines the syntax for certain adaptations and provides a set of **adaptation operations**. These adaptation operations are *generic* and pre-defined for each model element being part of information design, navigation design, presentation design, and operations design. In addition to these generic adaptation operations, additional **application-specific** adaptation operations can be defined by the customisation designer. For a detailed description of the generic customisation model, the reader is referred to [11] and [13].

## 6. Conclusions

The paper presents the overall UWA methodology and identifies four main aspects that belong to the model of a modern Web application: Requirements, Hypermedia elements, Customisation aspects, and Transactions.

Besides refining the methodology, our future work will be devoted to the implementation of a prototype environment, based on Rational Rose, and to the dissemination of the modelling approach by applying it to the specification of new challenging applications and by *convincing* new customers to adopt UWA for modelling their Web applications.

The adoption of the UWA methodology will imply advantages in terms of quality and time by tackling the main foreseen problems in the design of such applications: the definition of proper concepts and languages for describing their behaviour, the ability of tracking correlations among their different components, and the capability of correlating their requirements to design and implementation.

The ongoing design and prototype implementation of a B2B e-commerce application and an e-banking application, based upon the above methodology, are crucial to validate the quality and the effectiveness of the design approach (methods and tools), and its capability of being adequate to real-life environments.

Interested readers can refer to [14] for all details about the project.

**References**

[1] 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

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

[3] A. C.W. Finkelstein, A. Savigni, G. Kappel, W. Retschitzegger, E. Kimmerstorfer, W. Schwinger, Th. Hofer, B. Pröll, Ch. Feichtner, Ubiquitous Web Application Development - A Framework for Understanding, The 6th World Multiconference on Systemics, Cybernetics and Informatics, Orlando, Florida, US, July 2002

[4] 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

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

[6] L. Kleinrock, Nomadicity: Anytime, Anywhere In A Disconnected World. Mobile Networks and Applications, 1(4), Jan. 1996

[7] Object Management Group. Unified Modeling Language (UML) Specification. Version 1.4, Technical report, OMG, September 2001

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

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

[10] L. Baresi, F. Garzotto, and P. Paolini, Extending UML for Modeling Web Applications. In Proceedings of 34th Annual Hawaii International Conference on System Sci-ences (HICSS-34). IEEE Computer Society, 2001

[11] G. Kappel, W. Retschitzegger, W. Schwinger, Modeling Ubiquitous Web-Applications - The WUML Approach, Proceedings of the International Workshop on Data Semantics in Web Information Systems, Kyoto, Japan, 2001

[12] G. Kappel, W. Retschitzegger, W. Schwinger, Modeling Customizable Web Applications - A Requirement's Perspective, Proceedings of the International Conference on Digital Libraries, Kyoto, Japan, 2000

[13] G. Kappel, W. Retschitzegger, E. Kimmerstorfer, B. Pröll, W. Schwinger, Th. Hofer; Enabling Ubiquity for Web Applications - Customisation Modelling for Web Applications; submitted for publication

[14] UWA consortium. www.uwaproject.org