The UWA Approach to Modeling Ubiquitous Web Applications

UWA Consortium
Piazza del Carmine,22 – 09100 Cagliari (Italy)

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 UWA Consortium to propose a special purpose design approach to modeling Web applications. This paper introduces the approach and sketches the main design steps.

I. INTRODUCTION

Web applications have already evolved to distributed systems that exploit the Internet as communication means and the Web as interface to access services and data ([2]). Nowadays, they are facing a new transformation to supply users with ubiquitous systems that are available anytime, anywhere, and with any media ([10]).

These new applications supply the user with both data and services, are multi-channel, that is, available on a variety of devices, and are used by several classes of users, with different needs and expertise. Even if most of the features of these applications are not new, their combination implies a new multi-aspect modeling approach that cannot be obtained by just piling up existing methodologies, tools and techniques. Given this belief, the UWA consortium[1] started working on the special-purpose modeling methodology that is presented in this paper.

The overall modeling problem is partitioned in 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 transactional behavior exhibited by the application and how it is affected by multi-channel delivery;
- *Customization 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 modeling activity is defined in terms of a *metamodel*, which captures the set of relevant concepts

---

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

and the primitives, a *notation*, based on UML ([6]) to represent the concepts, a *set of guidelines and heuristic*s, 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. Unfortunately, in this paper space limits oblige us to simply introduce each modeling aspect. Interested readers can refer to [8] for an in-depth presentation.

The UWA project provides a *unified framework,* which integrates the various meta-models and notations and highlights their mutual interdependence, and a *unified software environment, based on Rational Rose,* that integrates the tools specific to each modeling activity.

The rest of this paper presents each modeling phase in detail: each section introduces the main concepts design heuristics. The last section concludes the paper and introduces the future work.

II. REQUIREMENTS ELICITATION

The major influence on our approach to requirements engineering is Axel van Lamsweerde's KAOS work [3]. Work by Lamsweerde [3] (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 briefly introduced in the following.

A **stakeholder** is someone or something that has an interest in the system. 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). 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. The centre of our world is no longer the system, but rather the stakeholders of the system. A goal delivers a certain **value** to its stakeholders. The value is extremely hard (and probably impossible in the general case) 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. Refining goals can also help identify new ones and better understand those already elicited. Refining a goal into subgoals helps identify **conflicts** between goals. Conflicts must be solved as soon as possible, and always before the operationalization step, that is, before any of the goals involved in the conflict are turned into actual requirements.

Requirements also have an associated **priority**. Prioritising requirements becomes very desirable in any realistic software engineering methodology. There comes a time when a designer realises he simply cannot implement all of the requirements at the same time or in the same version. 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.

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

- *content*: this is the core value of a Web application. Content refers to the set of ideas and messages that the site communicates to its users. Ideas and messages are mainly specified in terms of the core information objects available
- *structure of content*: requirements can give a coarse-grained insight into how the information objects identified are structured. By *structure* we mean the organisation of content within the same information object
- *access*: this dimension refers to the navigational paths available to the user to reach the needed content
- *navigation*: requirements can suggest a connection between different information objects allowing the user to navigate from one piece of content to another
- *presentation*: requirements can also give guidelines for defining the visual communication strategies for presenting content, navigational capabilities, and operation to the user
- *user operation*: user operations are those operations that are visible to users. They are the only operations the users must be aware of
- *system operation*: system operations are those operations that are not visible to users, but are essential in building user operations.

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). The number and nature of dimensions is not fixed, and new ones can be added at will and at any time.

III. HYPERMEDIA DESIGN

Hypermedia aspects in UWA are dealt with by suitably tailoring W2000 ([1]), whose main concepts are organized in three main models:

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 generalization hierarchy. An entity is organized in semantic sub-units, called **Components**, which are pure organizational 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 organized 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.

The **Navigation model** specifies the concepts that allow the designer to reorganize 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 contents is organized 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 renders. 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 characterized 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.

The **Presentation model** specifies the concepts to make the designer 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 granules at presentation level. They can either come from nodes or add new contents that are 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 Link**s, 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 a 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 impact on 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 (with respect to their execution) computational steps that can be invoked by the user, or could be part of activities. A simple operation 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 any more. A multi-step operation is constrained on its borders only, but suitable scenarios can be defined to explain the different steps through which the execution evolves.
- *Activities*, which are not atomic anymore. They can be seen as business transactions or/and containers for operations (both simple and multi-step ones). Activities identify sets of operations to which different behavioral semantics can be associated. For example, either the whole activity is seen as an atomic transaction, or other more sophisticated transactional properties could be associated with

the activity to better control the effects of its execution.

IV. TRANSACTION DESIGN

Transactions in web applications are critical for businesses. Web transactions can be complex, the may be composed of several sub-transactions, they may be accessing many different resources including existing legacy systems and they may have complex semantics. To deal with such complex applications, web transaction design needs to be very flexible allowing both developing web applications from scratch by decomposing user level goals into sub-goals that exhibit transactional behavior (top-down design), as well as using already existing systems or services to compose new applications offering added value services (bottom-up design).

Transaction models that provide for transactions with complex internal structure are known as extended transaction models (ETM) and up to now several different such models have been proposed (sagas, nested, open nested, etc). Some recent web standards have adopted and new proposals are continuously appearing. Although the ETMs are valuable in many application domains relaxing some of the ACID transactional properties, they can't always deal with the full complexity that some modern ubiquitous web applications have. Their limitations come mainly form their inflexibility to incorporate different transactional semantics in one (structured) transaction or to describe different behavioral patterns for different parts of the same transaction.

Our objective is to facilitate the complex design process for web transactions by providing a high level modeling language based on extensions of UML for designing complex web transactions In particular our objectives are to:

1. Provide a formal, high-level design mechanism for designing both the static structure of transactions and their dynamic behavior.
2. Provide the ability for designing transactions compatible to most of known transaction models.
3. Provide for designing transaction models for scratch. As new models may be needed according to the application's requirements the ability to define new transaction models becomes very important.
4. Provide for describing different transaction decomposition semantics and behavior in the same structured transaction. This is very important for applications that access resources with different interfaces, behavior and semantics. With this ability the same transaction can access different resources and utilize existing legacy systems or services adapting to their behavior.
5. Provide for modeling activities with weaker transactional semantic that they do not have all the ACID properties.

To achieve the above objectives we propose UTML (Unified Transaction Modeling Language) as a high level modeling mechanism for web transactions. The core of UTML is a transaction meta-model that is

flexible enough to describe complex transaction according to application's requirements.

An important model primitive of the meta-model is the concept of the *activity*. An activity is a set of operations or other activities, the execution of which has to obey specific constraints and semantics both on what they must satisfy and when they can be executed.

Each activity has an *execution contract* that has to obey. An execution contract is defined by a set of properties concerning the execution of an activity instance. Such properties are *Atomicity, Consistency, Isolation* and *Durability*. It should be noted that an activity can have any (with some exceptions) possible *PropertySet* from empty (no contract is defined for the execution of activity instances) to ACID (forming an execution contract similar to traditional transactions).

Another basic concept of this meta-model is the concept of *operation*. An operation is an atomic, not suspendable unit of work (*semantic operation*). Each activity has a set of operations (*OperatonSet*) that implement its logic (*Functional Operations*) or manage the activity itself (*Management Operations*). Management operations are classified as *Initialization* and *Termination* operations. Each activity may be decomposed into a number of sub-activities which compose its *ActivitySet*. By associating ActivitySets and ManagementSets with each activity separately, we can better specify its behavior incorporating different behavioral patterns into the same structured transaction.

The decomposition of an activity has been also modeled in terms of *vitality* and *visibility*. On this basis, each activity decomposition association has been enhanced with the properties vital/non-vital and visible/invisible.

Finally, the structural dependencies that can be defined in the Organization Model are supported by the notion of *well-formedness rules*. A well-formedness rule is a constraint that applies on a structured activity and can specify or formalize the message flow between activities, the commit or rollback process, etc. The syntax of well-formedness rules is based on the aforementioned concepts and can be described either mathematically or through OCL (Object Constraint Language) of UML, based on a specific UTML organization model. However, it should be noted that in order to express constraints on activity instances (real time behavior) appropriate concepts should be defined. Thus, we associated with each activity instance the concept of *OperationHistory* (OS) and *ActivityHistory* (AS). These sets contain operation instances and activity instances of all successfully executed operations and activities accordingly.

Like UML, UTML is extensible. This can be done by introducing new management operations to describe the transaction management and well-formedness rules to define the complete behavior of activities and the flow of messages that are required to be exchanged.

UTML is supported by a UML compatible notation. That is, a specific UML profile for transaction design. For this profile, appropriate UML stereotypes have been introduced, using the UML extensibility mechanism, to support both the Organization and Execution Model and a design tool is under development.

The main contributions of this meta-model are the following:

- It provides description for both structural and execution dependencies of transactions.
- It 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 behavior of different transaction models into the same transaction.
- It distinguishes between management operations and functional operations that a transaction has giving the ability to specify its behavior.
- It provides for designing transactions with execution contracts weaker than ACID integrating diverse resources like legacy systems. Moreover, it formalizes the decomposition of such transactions and the propagation of these properties in sub-transactions.
- It introduces the concept of well-formedness rules that are based on well-described concepts and are used to describe intra and inter-transaction dependencies. Well-formedness rules and management operations compose the extensibility mechanism of the meta-model enabling for describing application-specific transactional behaviors.
- It uses finite state machines to describe transaction execution flows and run time execution dependencies between transactions.
- It provides an extend UML based notation, with appropriate stereotypes, that is used to visualize and document the transaction design.

Currently, UTML is being extended to provide support for modeling ubiquitous transaction execution, i.e. description of transactions that can be executed on disconnected hosts and be synchronized with the central database later on. This includes modeling not only of the synchronization process but also of resource pre-allocation for transactions in order to be able to be executed in a disconnected mode. Another underway extension of UTML has to do with the capability of modeling persistent activities. This is important for execution of "long-lived" transaction, especially under unstable or weak connection. Finally, extension for modeling dataflow dependencies between transactions will give us the ability to describe scopes and strategies for activities.

V. CUSTOMIZATION DESIGN

The approach to customization design is based on a broad view on customization [4]. Although most often separated in existing approaches [5], we think that customization for ubiquitous web applications should uniformly consider *personalization* aspects, together with issues resulting from being *ubiquitous*, thus supporting the anytime/anywhere/anymedia paradigm. In our opinion, the design space of customization comprises the two orthogonal dimensions *context* and *adaptation*. The context dimension comprises the circumstances of consumption of a ubiquitous web

4

application mainly dealing with the question "why to customize 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 customization. The adaptation dimension mainly refers to the questions which changes to make as well as what to change. *Customization* is seen, in turn, as a combination between a certain context and certain adaptation, thus adapting the ubiquitous web application towards a certain context. In particular, customization is regarded as a new dimension, influencing all other tasks of ubiquitous web application design as described in the previous sections.

For designing the customization, we propose a *generic customization model* in the sense of an object-oriented framework, which provides the customization designer with appropriate model elements for specifying both context and adaptation. Generic means that the model is application independent and provides some pre-defined classes and language constructs in order to model application dependent customization. In addition, the pre-defined classes can be extended by the customization designer through sub-classing in order to cope with application specific details.

To support the context dimension, we define a *physical context model*, comprising a set of pre-defined context classes, holding actual, historical and future information about the environment of the application and the application itself, e.g. the device used, the user accessing the web application. Second, there is a *logical context model*, which contains a set of pre-defined profile classes for providing more abstract and static information about the context, e.g., descriptions of the properties of a certain device, user profile information. Third, the *customization rule model* allows to specify certain customizations. The adaptation desired towards a certain context is specified in terms of *customization rules* which are specified within UML annotations attached to those model elements being subject to customization. The customization 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 to specify 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 customization designer.

VI. AN EXAMPLE

This section presents an example for an interactive, Web-based tourist guide, designed with the UWA methodology. Within requirements' design, requirements are expressed as a

directed acyclic graph going from higher-level goals to requirements proper (the leaves of the graph) [2].

Figure 1 shows a partial derivation graph including functional (e.g. *Guide Tourists*) and non-functional (e.g. *Maximise Ubiquity*) requirements of ubiquity and customisability. Obviously, this does not in any way claim to be complete, but only serves the purpose of showing the underlying process.
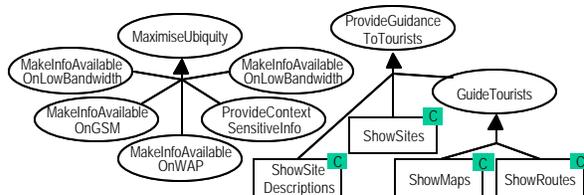
Figure 1. A (very partial) goal derivation graph

The example depicted in Figure 2 shows a small fraction of the tourist guide's Navigation Design, modelling the Navigation Cluster for a tourist site. The Navigation Cluster contains the available nodes, holding information about the site's description, i.e., a short description (cf. `TouristSight`), a full description of the tourist sight (cf. `FullDescription`), and a route information (cf. `RouteInfo`), as well as the links in-between.
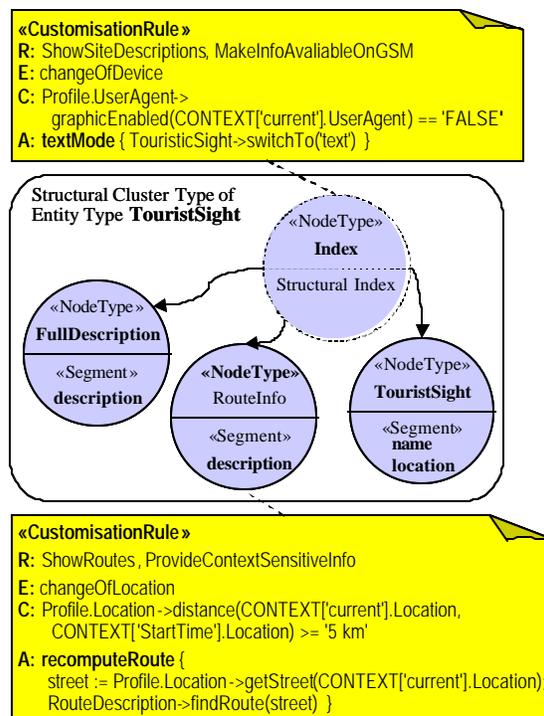

Figure 2. Partial Navigation and Customisation Design

The customisation rules are formulated in pseudo code and are attached to the W2000 elements. The specification of the requirement, which is realised by a customisation rules, is marked with `R:`. The `E:`, `C:`, and `A:` indicate the event, condition, and action of the customisation rule, respectively. The underlying

---

[2] For space reasons, only tiny snapshots of the actual graph will be shown.

physical and logical context models for this example are not shown due to limited space.

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`. 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.

VII. CONCLUSIONS AND FUTURE WORK

The paper presents the UWA approach to modeling ubiquitous Web applications. Space limits obliged us just to sketch the methodology, but interested readers can refer to [7] for all details about the project.
As future work, we are about to start the implementation of the supporting tools and use special-purpose case studies to assess and evaluate the soundness of the approach.

REFERENCES

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

[2] G. Booch. The Architecture of Web Applications, 2001. `www.developer.ibm.com/library/articles/booch_web.html`.

[3] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed Requirements Acquisition. *Science of Computer Programming*, 20:3–50, 1993.

[4] 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.

[5] 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.

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

[7] UWA consortium. `www.uwaproject.org`

[8] UWA Consortium. General Definition of the UWA Framework. Technical report EC IST UWA Project, 2001.

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

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