A UML Extension for Designing Usable User Experiences for Web Applications

Vito Perrone[1], Luca Mainetti[2], Paolo Paolini[1]

[1] HOC (Hypermedia Open Center), Politecnico di Milano, Italy
perrone@elet.polimi.it
[2] SET-Lab (Software Engineering and Telemedia),
Università degli Studi di Lecce, Italy
luca.mainetti@unile.it

Abstract

In this paper we introduce our framework for supporting the entire development of interaction and data intensive (typically Web) applications and describe one of the composing methods addressing the design of the user experience. Current proposals, both in the academic and industrial communities addressing such a kind of application, exhibit different weaknesses and strengths but are both characterized by poor acceptance by the current practice. Instead of proposing a new, richer modelling method, we have extracted and reused what good has been done in both the academic and industrial worlds in order to meet potential stakeholders' requirements. The whole approach has been shaped by the domain analysis and addresses the development of Web applications from requirements elicitation/analysis to software design in four phases. One of these phases, the user experience design named E-WOOD, is here detailed. Its specific stakeholders and requirements are here described. E-WOOD extends a UML proposal, coming from the industrial world, reusing web engineering principles coming from the academic experience. It introduces a reasoning oriented, user centered semantics which can be used for designing application better fitting stakeholders' goals and closer to final user expectations.

1. Introduction

Modern Web applications are assuming more and more a key role in most of the computer mediated human activities. ECommerce, eBanking, eFinancing, eLearning and so forth, are only some important fields where the success of the business is strictly related to the quality of the Web application acting as mediator towards final users. As far as shown in [1], in these applications user effectiveness and stakeholders' goals satisfaction are crucial for their quality. Typically, such applications provide users with a large amount of different information (data intensive applications) and integrate operations and business processes. As distinctive characteristics, all these services are accessed in a highly interactive way exploiting the navigational paradigm. Furthermore, currently services are more and more offered exploiting different delivery channels. These characteristics, together with the growing complexity of the delivered applications, have considerably increased the intrinsic complexity of such a software category.

Our assumption, quite agreed by the academic and industrial communities [2], is that quality for such applications is strictly related to a good design. As a consequence of the above considerations, it is clear that in this stage industry needs systematic design methods that could help in assuring the required quality [3].

Looking at the current methods addressing the design of Web applications, we discern two different communities. From the one side, we have the academic community, in particular the Web engineering one, where a considerable number of specific design methods have been proposed along the last decade [4-11]. Generally speaking and neglecting some peculiarities, most of these methods address the *conceptual design* [4] of Web applications. They are characterized by rich *semantics* coping with numerous peculiarities of Web applications. On the other side, there is the UML community where representative companies from the industrial world are employing a massive effort [17,18] in defining a reference modelling language for supporting the development process of software systems. UML native methods address the *logical design* [4] of an application, in that most of the modelling primitives abstract from concrete implementation artefacts. Being their modelling primitives closely related to concrete counterparts, these methods result easier to be understood and used by technicians as support for the implementation activities. On the contrary, when referring to the Web application domain, where the user interaction with the system plays a key role, it is recognized that UML support is quite vague [5]. It is mostly due to the fact UML lacks of a proper semantics which should help designers during the analysis activities to devise a user oriented application. Confirming this trend, the most recognized UML method proposed by the industrial community, that is, the WAE [12] introduce ah-hoc primitives for modelling typical software components of a Web applications, like *client page*, *server page*, *frameset*, etc. Evidently, these concepts do not belong to the user experiences so they cannot be used to reason on how to improve it.

Conversely, they can be effectively used to reason about the software architecture before coding it.

Besides their peculiar characteristics, the adopted modelling languages, primitives, strengths and weaknesses, both communities advocate the MDA as a way to improve the final product quality. However, they front this approach with a different philosophy. Academic proposals aim at designing what final users should perceive, carefully matching this design against the achieved requirements. Conversely, they often neglect architectural concerns. In other words, we can say they embrace the idea that quality is mostly decided at conceptual level so they typically pass from the conceptual design to the code. This position is also evident by the fact that most of these approaches propose support tools that can produce an application prototype (usually an evolutionary one) from the conceptual design. On the other hand, industrial approaches pay more attention to the correct design of the software modules that compose the system architecture often overcoming an accurate design of the user perspective. Most of them analyze user requirements by means of use case diagrams [18] describing which functionalities are supposed to be provided by the system to its users. Then, the user interaction with the system is detailed by means of sequences or collaboration diagrams [18] that describe the dynamic properties of these functionalities in some relevant scenario. On the basis of such an analysis and the chosen architecture, designers have to define the software that will meet the achieved requirements. In other words, we can say they embrace the idea that *quality can be assured by a good analysis and if suitable software models are crafted*.

Recent studies [3, 14, 15, 16] demonstrate that in the Web domain most of the current proposals have only slightly impacted on the actual practice. What are the reasons? Which of the above philosophies should be adopted to define a more suitable design model? Various can be the factors that hinder the adoption of systematic approaches for modeling.

In this light, the methodological framework we introduce in this paper aims at embodying the advantages of the two mentioned philosophies. It is composed by four phases embracing the web application development lifecycle from analysis to software specification. Each phase adopts a specific design model which has been defined on the basis of an accurate analysis of its stakeholders' goals. In particular, in this paper we describe our proposal for designing the user experience. This method, named E-WOOD, extends the Conallen's UML proposal for designing the user experience – UX [12] but embodies the semantics of a known academic method, called W2000 [25, 26], we are familiar to.

To illustrate vividly the approach and in particular the user experience modelling method, we will use real examples from the design of a running Web application which we have designed and developed: the Website "Munch und Berlin" (www.munchundberlin.org). It has been originally realized for the State Museum in Berlin within the HELP EU-funded project with the aim of providing to the general public (including users with visual disabilities) a Website promoting the temporary exhibition of Evard Munch's prints. Being a real, even if relatively small-sized, application, it is quite suitable to illustrate the main aspects of our approach.

2. Web application design: panorama and related works

Along the last ten years a number of methods have been proposed for supporting the design of Web applications. In the following of this section we briefly resume the main characteristics of these methods from two perspectives – the academia and the industry – and considering their role with respect to the analysis and software design phases. Looking at the academic community, some of the most known existing methodologies are HDM [6], W2000 [25, 26], OO-HDM [8], WebML [10], UWE [11], WSDM [9], OO-H [13], etc. Roughly speaking, they specify the design of a Web application at the conceptual level, neglecting technological aspects and constraints. Besides technical (minor) differences, these methods share lots of common features. All of them are based upon an information-navigation paradigm to describe the user interaction, recognize the importance of the semantics as guidance for conceiving the application design and share the fundamental principle of *separation of concerns.* On the other hand, they differ one with another in terms of proposed design primitives, notation and support tolls. All these methods Following this principle, and adopting the W2000 [25, 26] terminology[1], the design of a Web application is achieved in four dimensions: *Information and Access Structures design*, defining the basic conceptual information units (entities) as perceived by the user and the navigational infrastructure in terms of semantics associations (between related entities) and

[1] In this paper we use W2000 as example of academic design method since it has been developed in our research group so we are very familiar to its terminology. Nevertheless, we are firmly convinced all our considerations are quite independent from it and generic with the respect of other similar design methods.

access structures (navigational paths enabling users reach interesting information units); *Operations and Business Process design*, defining operations (e.g. “add to shopping cart”) and processes (e.g. “check-out”, “registration”) within a Web application; *Navigation design*, defining the navigation network allowing users browse information and access structures and execute operations and processes; *Presentation design*, defining the page structure in terms of lay-out aspects and graphical elements and the page organization and navigation.

Although, if properly used, current academic methods have the potentiality of enabling designers conceive high quality (say usable and effective) applications, they suffer, as stated in a recent study [3], of some inefficiencies which contribute to a poor acceptance from the industrial environment. Owning sophisticated and semantically rich primitives often it *takes too much effort and time in order to learn and start using the methods*. *Modelling purpose is only badly or vaguely specified* with the respect of the overall development process. It is often claimed models are intended as support tool during the early analysis activities, but they then their models are also used to automatically generate the running application [13], [10]. *Cumbersome design documents* are generally produced as output of the design activities. These documents risk being hard to read and use both during the analysis and the following implementation activities. *Proprietary concepts and notation* are generally proposed (except a few cases like [11]) by each method increasing the learning time and thus the negative perception of industry people [21]. *Ad-hoc and in-house made support tools* are generally proposed instead of commercial ones.

With regard to the second category, that is, methods proposed in by the industrial world, UML is definitively considered the standard de-facto in the design practice. Referring to the web application domain, the only recognized method coming from the industrial environment is the one proposed by Conallen in [12],[20], that is, the Web Application Extension (WAE). WAE, like other UML native methods, adopts an implementation oriented approach, in that most of the modelling primitives abstract from concrete implementation artefacts. Examples of WAE primitives are *client page, sever page, style sheet*, *frameset*, etc, obtained by stereotyping UML classes and *link, redirect, submit*, etc., obtained stereotyping UML associations. Due to this characteristic, they are quite easy to understand and use by technicians for supporting the software design activities and broadly supported by commercial tools. On the other hand, concerning WEB applications, it is known [5] that UML lacks of proper semantics for supporting the design of communication and navigation aspects both during the analysis and design phases.

Finally, the topic of explicitly considering stakeholders and their requirements for shaping a suitable design method has been barely fronted by existing approaches. In most of examined literature when a new modelling method is proposed, the well-known and high level software engineering principles are, at most, cited. For example in [5] it is argued that the next generation of OO methods “*…should be sufficiently user-friendly to all kinds of possible stakeholders. That is, for all stakeholders of any model, its relevant parts expressed in the modeling language, must be understandable, must be clear even. For the modeler as well as for all other persons involved in the modeling activity, any model must be expressive, precise and clear as well*”. However, besides these well known software engineering principles, we also advocate that, due to the diversity of all possible stakeholders, the lack of an explicit consideration of what every potential stakeholder expects by the modeling method could be one of the main reasons of the existing gap between current proposals and industry practice.

3. Analyzing Stakeholders’ Requirements

To be successful, design methods, as well as any engineering product, should accomplish the needs and expectations of its potential stakeholders. Defining a new method requires an accurate analysis of goals and requirements of their users, i.e. the practitioners who daily conceive, develop and deploy applications, and other potential stakeholders whose needs may influence the method definition. Neglecting stakeholders’ needs can bring to lack of attention towards these engineering products (design models) by the industrial practice while fitness to requirements can drastically increase their acceptability at wider level. On this basis, we have defined our approach by taking explicitly into account its potential stakeholders. It is composed by four phases embracing the web application development lifecycle from analysis to software specification. Each phase adopts a specific design model which has been defined on the basis of an accurate analysis of its stakeholders’ goals. In this paper we focus on the conceptual design of the user experience which is usually achieved between the analysis activities and the software design.

3.1 Requirements for a conceptual model addressing the user experience design

Conceptual models are used at the beginning of the overall design activities, as intended in the software engineering discipline, which will finally lead to the detailed specification of the software modules to be coded. In this phase, the main goal of conceptual models is to clearly define the solution (application to-be) characteristics, even if still avoiding implementation details. In the following potential stakeholders (the most relevant ones) and their relative requirements, gathered in our experience on the field, are described. It should be noticed that not all the described stakeholders are also active users of design method, but their needs can indirectly influence the method definition.

Designers: are in charge of the system design. Depending on the reference community, the terminology adopted within a company, the kind of application, and so on, different professional figures (e.g. information architects, interaction and usability experts, and so on) might be attributed to play this role. Usually several designers work both in the analysis and design phases thus first goal is to ***ease the communication with the analysis activities and among different designers in the design activities***. For the former, some form of *guidance* should be provided *to support the passage from the early solution devised in the analysis activities to the actual design* of the system. This *mapping should compromise between rigour* – to enable some form of automatic passage – *and flexibility* – to not constraint choices designers have to perform in the design phase. In this phase, they have to design models very close to the application to-be, thus inevitably these models are rich in details and the specification is often composed by several heterogeneous diagrams representing different application concerns. To master the overall design complexity (avoiding naive designers feel lost) the method should *provide an explicit framing strategy*. Furthermore, model drawing is time-consuming activity that needs proper tool support. In order to be used in professional environments, *support tools should adhere to the commercial standards*. Since building such tools is an expensive activity, new modeling methods should be defined so that *existing commercial tools can be exploited*.

Usability experts and Graphical designers***:*** depending on project parameters like those mentioned above, these roles could be attributed to designers or other professionals with non technical skills. However, in WEB applications these aspects are taking more and more importance and require specific competences. Whatever is the case, these figures are interested in carefully ***defining and reviewing usability and graphical aspects*** of the application to-be, thus *concerns impacting usability and layout/graphical aspects should be explicitly modeled and made easy to access*. These experts are used to analyze and discuss about usability and graphical concerns by means of mock-up or other similar representations that closely reproduce the application to be. Thus, to achieve an effective communication with usability and graphical experts, *models should* also *look as close as possible to the actual application*.

Software designers and Implementers***:*** define and implement the software modules that will actually realize, on the basis of the chosen system architecture, the application specified by the conceptual models. From our experience on the field, a recognized lack of ***existing conceptual models*** is that they ***require a considerable effort to be mapped into software artifacts***. Often, it is hard to understand which diagrams should be considered for obtaining a single software artifact and, most of times, several different diagrams must be composed. For example in the web domain, to design a server page, software designers have to refer to information models for the page data, operation and business process models for the business logic, navigation models for the navigation logic and presentation models for graphical and layout aspects. Software designers consider this activity being time consuming and, if not properly supported by tools, a possible source of mapping mistakes. On the basis of these considerations, models should *embody modeling primitives as closer as possible to concrete counterparts* and that *as less as possible diagrams should be considered to define a software component*. Also the *design documentation to be used for supporting the implementation activities should be concise and easy to read* (many cross-references among different diagrams are considered highly annoying). Another highly desirable feature a modeling method should own, for these stakeholder types, is to *provide predefined mapping strategies – let's say mapping patterns – towards the most known architectural patterns*. Finally, most of the interviewed software designers and implementers were already used to the UML and related CASE tools, thus they showed a remarkable preference in having conceptual models described in UML-like notation and following the UML philosophy, that is, modeling methods should *belong to the UML family.*

Product manager: this stakeholder type represents the most important client counterpart dealing with the application design, and act as interface of decision makers, opinion makers, clients and content/domain experts. Product managers are usually in charge of assuring the envisioned application will be able to satisfy the client company expectations, but they also are responsible of a number of other specific tasks. Among others, one the most important is to set up the editorial chain. Their main, somehow opposite, goals are *to take the control of the overall application at a glance* and *to get details of specific aspects (related to their tasks)*. Desirable features for the method should be to *review models at different levels of detail*, to *embody most of the needed information to set up the editorial chain* and to *enable some form of requirements tracking*.

Final Users***:*** this stakeholder category is the more important for tuning the application interaction even if it is also the less accessible for several reasons. In fact, they usually are not part of the client, are barely identifiable and their characteristics can vary remarkably. Nevertheless, gathering some feedback from potential users before the coding activities start can bring several advantages since modifying models is much less expensive than modifying code. From our experience [27], a discussion with users mediated by models is usually ineffective because they need to see and handle application as it were running. Application prototypes are much more effective in this development stage, thus *models should be easy to turn into prototypes.*

Testers and Evaluators*:* models produced in the design phase are also used by testers and evaluators once the application has been implemented. In these phases, models should provide the ground for setting up the testing or evaluation plan. Testers and evaluators need *different concerns to be evaluated being easily identifiable* in the implemented application. Moreover, *models should look very close to the implemented application* so that testers and evaluators can easily match the running product to the originating models.

Table 1. Requirements for a conceptual tool for the design phase.

Stakeholders	*Design Requirements*
Designers	*R1. provide guidance for passing from early solutions to actual design* *R2. compromise between rigor and flexibility* *R3. provide a framing strategy* *R4. enable to exploit existing commercial tools*
Usability and Graphical experts	*R5. distinguish and make easily accessible concerns impacting usability and layout/graphic* *R6. models should look as close as possible to the actual application*
Software designers and Implementers	*R7. modeling primitives as closer as possible to concrete counterparts* *R8. as less as possible diagrams should be considered to define a software component* *R9. Concise and easy to read specification documents* *R10. predefined mapping strategies towards the most known architectural patterns* *R11. belong to the UML family*
Product manager	*R12. review models at different levels of detail* *R13. embody information to set up the editorial chain* *R14. enable requirements tracking*
Final Users	*R15. models easy to turn into prototypes*
Testers or Evaluators	*R16. different concerns to be evaluated being easily identifiable* *R17. models should look very close to the implemented application*

4. The whole framework at a glance

In this section we briefly introduce the whole methodological framework to better contextualize the proposed conceptual modelling method. In all the section, we specify precise references to the requirements discussed above as it becomes necessary.

In Figure 1 the composing phases are shown. A different modelling method is proposed for each of them. As well as other software development processes, we assume that these phases should be executed in an iterative and incremental way, therefore the picture only purpose is to express the phases order within the whole process. Considering the entire development process of a web application, we can say the framework covers both the *analysis* and *design activities* [28]. Moreover, adopting the Jackson terminology [22], we distinguish between the *problem* and the *solution domains*. These dimensions, the process and the domain, are used to organize the following discussion.

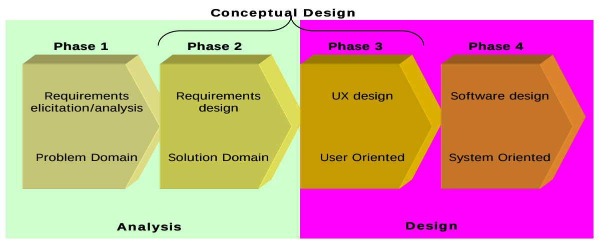


Figure 1: Phases in the development process of Web applications

The two left more phases are both achieved during the analysis activities. For supporting the **requirements elicitation and analysis** (phase 1) we propose AWARE [1], a goal-oriented method specially suited for web application requirements engineering. AWARE primitives include *goals* and *requirements* which definitively belong to the problem domain.

However, in our experience, discussing with stakeholders (analysis) about needs and goals can be too abstract for a fruitful reasoning about relative importance of various goals and requirements and for eliciting new ones [27]. A first very high level solution, focusing on specific topics, can help validation and elicitation activities (e.g. interviews) enabling a more concrete discussion about the problem. We call this activity **Requirements Design** (phase 2) meaning that in this phase requirements take a more concrete form accomplishing a preliminary hop from the problem domain to the solution one. In this phase we use IDM [24]. IDM (Interactive Dialogue Model) is a design model for interactive applications based on linguistic concepts of human dialogue. It bases on the interpretation of the interaction between the user and the application as a sort of dialogue. It is simple to grasp, and effective in representing the most relevant features of the application in terms of content of the dialogue and dialogue moves. In fact, three simple design elements characterize IDM: “topic”, “change of topic”, and “group of topic”. An interactive application may describe a “topic” (e.g. a “print”, or a “technique”); or it may allow the user to switch to a “related topic” (e.g. switching from a “print” to the “technique” used for it); or it may allow the user to start from a “group of topics” (e.g. “the masterpieces”, or “the prints dealing with sickness”) and then browse within the group.

Although in traditional SE approaches requirements are directly used for designing the software architecture (e.g. class diagrams, component diagrams, etc. using the UML terminology), in applications where the user interaction and the communication potential play crucial roles, the software design has to be postponed to the *user experience* design [12]. In this phase the application is designed as perceived by final users, neglecting how the software will be realized. Here, designers have to precisely define how users interact with the application to accomplish their tasks, taking care of the application usability and effectiveness with the respect of user requirements and quality expectations. In our framework, we achieve the concrete passage into the design phase by translating IDM models (phase 2) into E-WOOD ones (phase 3). IDM and E-WOOD, together, build up our approach to the conceptual design of WEB applications. Both methods take their foundations in W2000 [25, 26], last heir of HDM [6] recognized as one of the first conceptual methods for web application design. As described in section 2, W2000, as well as other similar conceptual models, implements the *separation of concerns* principle by structuring the design in four dimensions. Both our methods keep this principle at the basis of their definitions but projecting the previous dimensions in a sole dimension for the sake of conciseness, for reducing the number of concepts to be learnt and references among diagrams (R8,R9). The last step (phase 4) consists of a detailed design of the software that will be implemented to *realize* the desired user experience. This is generally called *logical design* of the system to-be. Passing from phase 3 to phase 4, a paradigm shift is achieved since, in phase 4, designers have to design the *system* that will *realize* the modelled user experiences. This passage is far to be straightforward and a number of trade-offs with the architectural constraints and various decisions have to be undertaken [27]. Models produced in this phase should specify a design easy to code. Here, we adopt the modelling method proposed by Conallen, namely WAE [12]. Our choice has been driven by two main reasons. First, it is already recognized in the industrial environment as the UML method for designing the software for web applications and a number of CASE tools already support its diagram drawing (e.g. Rational Rose, MS Visio). Second, as shown in paragraph 4.2, it is very easy and intuitive mapping WAE models upon E-WOOD as far as most of times, only one E-WOOD artefact is needed to define a set of related WAE artefacts (R8,R9).

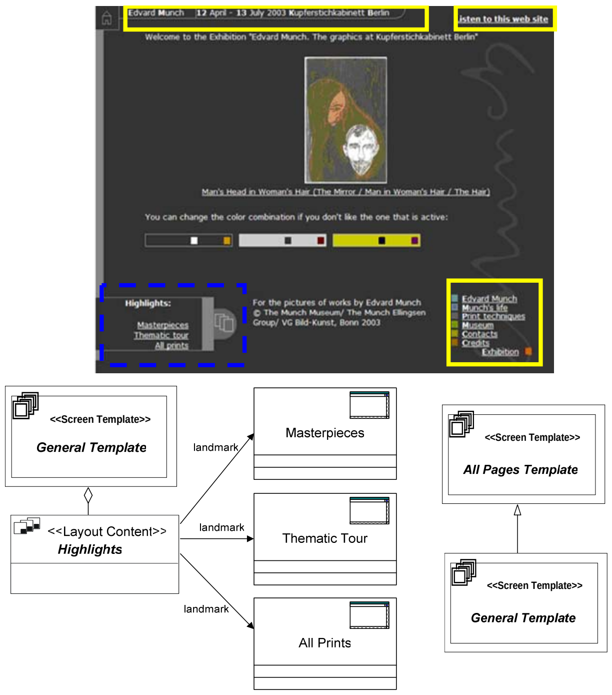
Finally, the methodological framework also includes a number of guidelines on how to use every method within each phase and how to move forward and back between adjoining phases. Guidelines are informally described in terms of patterns [29] so providing an useful but flexible guidance (R1,R2). They also front specific design issues like the multi-user and multi-channel design. Lack of space prevents us to describe this aspect, but the complete set of guidelines can be found in [27].

5. E-WOOD: the user experience design

Our proposal for designing the user experience, called E-WOOD, has been defined as a UML extension. UML has been chosen as modelling language to meet R11, while the extension mechanism has been preferred to defining a metamodel in order to exploit easily existing commercial tools (R4). Our model extends an existing proposal for designing the user experience, that is, the UX [12] since, as shown in the Conallen’s book, mapping WAE models upon UX ones is easy and intuitive (R8,R10,R17). UX’s high level primitives are *screen* and *links*, and an application is merely considered as made up of a number of screens connected by links. Typically, a set of WAE artefacts are mapped upon a screen by means of *realization* associations (stereotyped as <<build>>), specifying which logical elements (WAE models) build the various parts of the screen (contents and links). Our main goal in extending the UX has been to add the needed semantics (extracted by the W2000 primitives) to enable the separation of concerns impacting the application usability, its functionalities and the whole quality (R5,R13,R16). In E-WOOD different concerns are specified in different views and by introducing specific design concepts. These concepts have been defined extending standard *class* and *association* elements in terms of *stereotype, semantic description, constraints*, *tags* properties. An additional property (*mapping constraints*) has been also introduced to specify mapping constraints between IDM and E-WOOD models (R1,R2). As well as in UX, E-WOOD high level primitives are *screens* and *links*. Screens can aggregate both content and input forms; links can be used to perform a simple navigation among pages or to provide inputs to operations and processes. E-WOOD models are thus very close to the application to-be (R6, R7, R17) and easy to turn into prototypes or mock-ups (R15). Keeping these basic primitives we have also preserved the proven mapping capabilities towards the WAE (R10, R8).

The introduced semantics is also used to define a framing strategy (R3) which helps designers organize the overall design activities, fosters reuse and make design documentation more readable (R9). The framing strategy mostly reflects the W2000's design dimensions. E-WOOD proposes to organize the design of the overall application in five views. Each view includes several diagrams and makes use of specific stereotyped classes. Due to the lack of space, in the following we only introduce the main views to show the philosophy behind our method and how we have tried to accomplish the above stated requirements. The complete specification can be found in [27].

The *Template View* is used to define common contents and links of page sets. Examples of common contents could be the copyright information, the company logo and so on, whilst examples of common links could be those connecting to the home page or to the various site's sections (like those on the bottom of many web sites). Typically the template design involves the graphical designers who are in charge of the application look-and-fell (R5). The basic primitive used in these diagrams is the <<Screen Template>>, an abstract class used as place-holder for content and links belonging to a set of screens. Layout contents (both information and graphical elements) and common links are modelled respectively by means of <<Layout Content>> and <<Landmark link>> primitives. In Figure 2 a Web page of Munch is shown together with design excerpts taken from the template view.



Welcome to the Exhibition "Edvard Munch. The graphics at Kupferstichkabinett Berlin"

You can change the color combination if you don't like the one that is active:

For the pictures of works by Edvard Munch © The Munch Museum/ The Munch Ellingsen Group/ VG Bild-Kunst, Bonn 2003

Figure 2: Some excerpts from the Template view

Every Munch's Web page includes contents and links highlighted in the picture by continue line rectangles, while only some pages include also the set of links highlighted by a broken line rectangle. To model this we use two *<<Screen Template>>* abstract classes modelling the two different templates. The diagram on the bottom right corner shows that a specialization relationship is used to specify the hierarchy between these templates. The layout content belonging to the "General Template" is represented by the <<Layout Content>> class aggregated to the template, while the outgoing links are represented by means of the stereotyped associations <<Landmark>> ending on the target pages. It can be noticed as <<Screen Template>> is modelled as abstract class since it is defined only for generalizing content and links belonging to a set of concrete pages. A specific constraint is provided in the formal specification of the UML extension.

The *Structural View* is used to define pages enabling users explore information concerning the domain entities or IDM' topics. <<Content>> classes are aggregated to screen classes and models portions of the whole topic information. <<Structural link>>s are used to model the navigation achieved across pages belonging to the same topic. For example, as depicted in **Figure 3**, the overall information concerning the "Print" entity are organized in three pages (Introduction, Big Image and Description) which are connected by means of bi-directional links originating from the "Introduction" page. Each IDM topic is mapped on a number of content classes (and relative pages) equivalent to the number of its dialog acts. Content classes are then enriched by a fine-grain definition of data slots which can be used as input for setting up the editorial chain (R13). Content classes contain a Boolean tagged value called *entry point* whose purpose is to specify whether that portion of the content can be used as starting point for exploring the entity information. Following our framework guidelines, such pages should include, at least, a minimal set of entity attributes that can be used by the user to understand what the entity instance talks about. Information organization, kind of navigation and entry points are concerns usually discussed with communication and usability experts (R5,R16) taking in mind that when users navigate these pages are clearly interested in improving their knowledge about the entity.

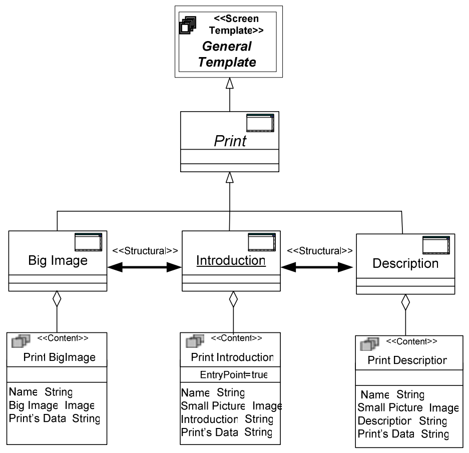


Figure 3: a) Structural view for the "Print" topic;

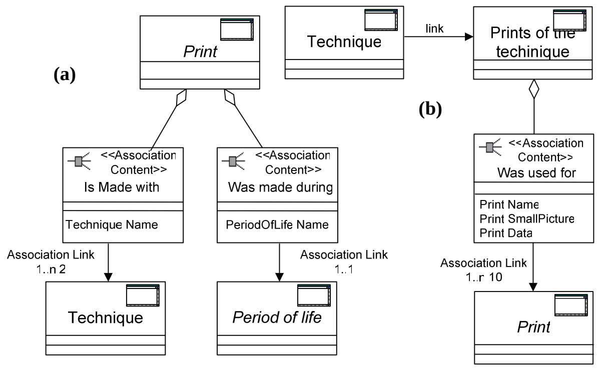


Figure 4: Association views

In the *Association View* designers specify how to pass from a discovered interesting topic to a related one (*relevant relation* in IDM). For this part of the user experience design, it is very important to carefully decide how to enable users to understand which of the possible target topic instances they are interested in. This aspect is called, in the HCI community, *information scent* and is one of the factors strongly impacting the application usability. In E-WOOD we use to this purpose the <<Association Content>> (**Figure 4** (a) and (b)). In (a) these information are integrated in all the "Print"'s pages (it is aggregated to the abstract page representing the common features of all the structural pages) and <<Association link>>s connect these pages to the target one. In (b), the "Technique" page includes a <<Link>> association which brings to new page "Prints of the Technique" whose only purpose is to list the possible target "Prints" which have been produced using the source "Technique". From this page a <<Association link>> point to the destination pages. The <<Association link>> primitive includes a tagged value that specifies the association *multiplicity* in terms of *min*, *max* and *expected* values. In particular, the expected multiplicity provides a useful indication about how many instances of the target entity are in general addressed by the association. This information can be used for taking some design choices like attaching the <<Association content>> to the source page or defining a new ad-hoc page (the two possible solutions shown above). Having *max* or *expected* cardinality very small, our guidelines suggest aggregating the <<Association Content>> to the source pages, while in case the expected number grows up, we suggest the other solution.

In the *Access Path View* designers have to specify the navigational paths enabling users find interesting objects. For example in an e-commerce web site like Amazon, examples of predefined navigational paths are the books categories which organize books by topic and sub-topics, but also a "Bestsellers" or personalized "Book recommendations" and so forth. The purpose of these pages is supporting users while exploring the proposed site content organization improving the user understanding. Such pages should help users in deciding how to move around possible choices enabling them exploring in depth the navigational path. In each path step, possible users should choice how to refine the set of possible interesting kinds of topics or, in case of terminal steps, which topic instance is worth to be examined (passing to the structural navigation) among the possible ones. Navigational paths are related to the IDM "Group of topics" concept. In the example in **Figure 5**, the E-WOOD model of a one-step path enabling users access to the most famous Munch's prints is depicted. The access structure is available in the page modelled by the <<Screen>> "MasterPieces". Here users can find an "Introduction" to the collection and a list of prints. For each print a short preview is provided by means of three print's attributes: "Small picture", "Name" and "Print's data". This information is modelled by the <<Access Content>> class aggregated to the <<Screen>> "MasterPieces". By means of these previews users can choice which print they are interested in and navigate to it by means of the <<Collection Link>> "Index". Once users land to the choose print page, he can also move back and forward among the collection members (other MasterPieces prints). To model this, in this diagram the <<Collection Link>> "Next/Previous Masterpieces" is added to the abstract <<Screen>> "Print". It represents the E-WOOD model for a case of *guided tour* pattern. It is important to be noticed as these links are only available in the context of this collection, so if the user reaches a print by other access mechanisms (other access paths, associations, search engine, etc.) he cannot move among prints contained in this collection. Access paths usually define a *navigational context*, in that new content and links can be added to entity pages when accessed by a specific access path.

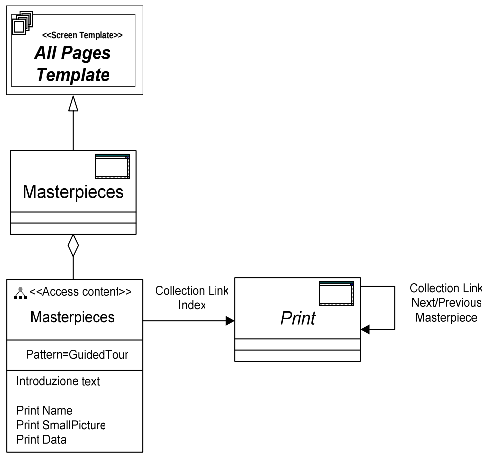


Figure 5: An excerpt of the Access Path View for the Masterpieces page

Besides these main views, we also propose a *Navigational Map View* that summarizes the main navigational features of the entire application. Our guidelines suggest how to choice candidate pages, among the overall defined in other views, to be included in the navigational map. Switching from the navigational map to the detailed design of contained screens it allows R12 being accomplished. Finally, the *Operation/Process View* complements the previous views adding to pages concerns related to operations invocation (e.g. "add to shopping cart") and defining pages involved in business processes execution. Lack of space prevents us to describe this complementary view.

6. Conclusions

We all know that the existing literature about conceptual methods addressing the design of Web applications is (over)abundant. On the other hand, we also know that a remarkable gap between theory and practice still exists [14, 15, 16],[3]. *Which are the reasons behind the poor acceptance by the practitioners*? Starting from these considerations, in this paper we have claimed that a possible reason might be that existing proposals have failed short by neglecting stakeholders' goals and expectations. In this light and focusing on the development of WEB applications, we have carefully analyzed the environment where a design method should operate identifying which are the potential stakeholder types and their goals and requirements. On the basis of this analysis, instead of inventing new design methods, we have reused or extended the best, in our view, of current approaches both in the academic and industrial communities. The approach covers both analysis and design activities and consists of four phases, executed in an iterative and incremental way. Defining it, we put in practice most of our experience achieved working on the field with conceptual design methods for Web applications [3], [27].

This paper has focused on our proposal for the user experience design, namely E-EWOOD. It is a UML profile that enables to specify the user experience in terms of pages and links but that embodies semantics enabling designers reason together with different stakeholder types about crucial concerns heavily impacting the application usability and effectiveness, that is, its perceived quality. Moreover, due to its definition, E-WOOD can exploit existing commercial tools for supporting the model drawing and perfectly match an existing and already affirmed, among practitioners, method for designing the software modules of a WEB application.

The approach has been applied in several design and reverse design case studies and industrial projects. Its transferability in industrial environments has been also experimented in two projects in cooperation with two Italian software companies (in the context of the GENESIS-D projects [27]). From these first experiences a number of considerations can be drawn out. Compared to W2000, we have noticed a significant decrease of the required learning time. Practitioners were able to use both methods after a short but intensive course (2-3 days). They drew IDM models using paper and pencil, while used VISIO™ stencils for designing E-WOOD and WAE models. In all the achieved experiences, we spent, with E-WOOD, on the average one third of the time required by W2000 to produce the same level of detail in the specification of several application designs. This has to be summed to the time required for manually drawing IDM models which is, however, very affordable. Compared to UX, we obtained several advantages mostly due to the introduced semantics. Models are more expressive and easy to be revisited; the framing strategy enables a suitable organization of the overall design activities; a number of well know design patterns, developed in the web engineering community, can be exploited to produce quality applications.

Finally, concerning future works, we are working in two main directions: (i) enriching the framework with guidelines and patterns for fronting specific aspects like the multi-channel design and the mapping of E-WOOD models upon the most known software architectures (JAVA and MS.NET); (ii) Concerning the second point, as said above, one of the reasons which guided our decisions in defining methods in phase 3 and in adopting WAE in phase 4 has been the reuse of commercial CASE tools already widespread in the industrial environment like MS Visio® and IBM Rational Rose®. So doing, companies already accustomed to these tools can easily step towards the adoption of our methods only adding a few stereotypes and our views strategy. Existing commercial tools do not support the design of models of phase 1 and 2. In order to improve the coverage of the whole approach with proper tools, we are already working for defining an ECLIPSE [30] add-in which should include, besides all the modelling primitives, also a set of semi-automatic rules for passing from a phase to the next one, some tracking mechanism among phases and a loose consistency check option. For example, the consistency manager could check if the several user views can be actually derived by the unique database and if the logic perceived by users when executing processes is compatible with the business processes implemented at the logic level.

10. References

1. Bolchini, D., Paolini, P. Goal-Driven Requirements Analysis for Hypermedia-intensive Web Applications. Requirements Engineering Journal, Special Issue RE03, Springer 2003.
2. Grady Booch. Language Once Was Key – Now It's Design. Windows Server System Magazine. February 2003 Issue.
3. Garzotto, F., Perrone, V. On the Acceptability of Conceptual Design Models for Web Applications. In Pro. of ER'03 Workshops, (IWCMQ'03), October 2003 , Chicago, USA.
4. Mylopoulos, J., Information Modeling in the Time of the Revolution. Information Systems, Vol. 23, 1998
5. Engels, G., Groenewegen, L. Object-oriented modeling: a roadmap. In A. Finkelstein, editor, "The Future of Software Engineering", Special Volume published in conjunction with ICSE 2000, 2000.
6. Garzotto F., Paolini P. HDM- A Model-Based Approach to Hypertext Application Design. In ACM Transactions on Information Systems, Vol. 11, No1 January 1993, p1-26.
7. Isakowitz T, Stohr EA., Balasubramanian P. (1995) RMM: A design Methodology for Structured Hypermedia Design. In Communications of the ACM Vol.38 No8 August pp 34-44.
8. Schwabe, D., Rossi, G. An Object Oriented Approach to Web-Based Application Design. Theory and Practice of Object Systems, 4 (4), J. Wiley, 1998
9. De Troyer, O., Leune, C., WSDM: a User-Centered Design Method for Web Sites, in Proceedings 7th International Wolrd Wide Web Conference, Brisbane, 1997.
10. Ceri, S., Fraternali, P., Bongio, A. et al., Designing Data-intensive Web Applications, Morgan Kaufmann, 2002.
11. Hennicker, R., Koch, N. A UML-based Methodology for Hypermedia Design. In volume 1939 of LN in Computer Science, York, England, October 2000. Springer Verlag.
12. Conallen, J. Building Web Applications with UML (s.e.), Addison-Wesley, 2003.
13. Gómez, J., Cachero, C., Pastor, O., Conceptual Modeling of Device-Independent Web Applications, IEEE Multimedia, April-June 2001 (Vol. 8, No. 2).
14. Barry and Lang: A Survey of Multimedia and Web Development Techniques and Methodology Usage. IEEE Multimedia, April-June, 2001
15. C. Britton et al.: A Survey of Current Practice in the Development of Multimedia Systems. Information and Software Technology, vol. 39, no. 10, 1997, pp. 695-705.
16. B. Fitzgerald: An Investigation of the Use of Systems Development Methodologies in Practice. Fourth European Conf. Information Systems, Lisbon, Portugal, 1996
17. OMG, Object Management Group: UML 2001: a Standardization Odyssey, October 1999.
18. OMG, Object Management Group: Unified Modeling Language (UML), version 1.5 (formal/04-04-04)
19. SDTimes Magazine. UML Adoption Making Strong Progress. August 15, 2004
20. Conallen, J., Modeling Web Application Architectures with UML. Communications of the ACM, 1999
21. Kaindl, H., et al. Requirements Engineering and Technology Transfer: Obstacles, Incentives and Improvement Agenda. Requirement Engineering journal 7(3): 113-123 (2002)
22. Jackson, M., The World and the Machine. Keynote Address at ICSE-17; in Proceedings of ICSE-17; ACM Press, 1995.
23. Yu, E., Modeling Organizations for Information Systems Requirements Engineering. Proc. of the 1st Int. Symposium on Requirements Engineering, RE'93, San Jose, USA, 1993.
24. Bolchini, D., Paolini, P., Dialogue-based Design for Multichannel Interactions. In Proc. of IWWOST04 workshop held in conjunction with ICWE'04, München, Germany.
25. Baresi, L., Garzotto, F., Paolini, P. From Web Sites to Web Applications: New Issues for Conceptual Modelling. In Proc. WWW Conceptual Modeling Conf, October, 2000.

26. 24. Baresi L., Garzotto, F., Paolini, P., Perrone, V. Hypermedia and Operation Design. Deliverable D7, European IST project UWA, www.uwa-project.org
27. Perrone, V., Bolchini, D., Designing Communication Intensive Web Applications: Experience and Lessons from a Real Case. In proc. of WER 2004, 9-10 Dec. 2004 - Tandil, Argentina. To appear in a special issue on Req. Engineering of the Journal of Computer Science & Technology, autumn 2005.
28. Ghezzi, C., Jazayeri, M., Mandrioli, D.: Fundamentals of Software Engineering. Prentice-Hall, 1991.
29. Gamma, E, Helm, R., Johnson, R. and Vlissides, J. Design Patterns: Elements of Reusable Software Architec-ture. Addison-Wesley, 1995.
30. Balconi, A., Mainetti, L., Paolini, P. Perrone, V.: GENESIS-D: Formal specification of the conceptual and logical models. Politecnico of Milan, deliverable D2.2, project Genesis-D (October 2004). Available on https://www.elet.polimi.it/upload/perrone/D22ModelloConcettuale.pdf in Italian.
31. Eclipse consortium. Eclipse – Home page. www.eclipse.org/.