BANCO: a Web Architecture Supporting Unwitting End-User Development

Barbara Rita Barricelli, Andrea Marcante, Piero Mussio, Loredana Parasiliti Provenza, Stefano Valtolina
Università degli Studi di Milano
Via Comelico, 39/41
20135 Milano, Italy
+39 02 50316290

{barricelli, marcante, mussio, parasili, valtolin}@dico.unimi.it

Giuseppe Fresta
Consiglio Nazionale delle Ricerche
Via G. Moruzzi, 1
56124 Pisa, Italy
+39 050 3152933

giuseppe.fresta@isti.cnr.it

ABSTRACT

Today end users are no longer mere consumers of computer tools but increasingly need to be more active producers of information and software artifacts. New techniques for engineering software are needed to support end users in this new role. This paper introduces one of these techniques, namely unwitting end-user development, and explains the BANCO architecture, which has been designed to support unwitting end-user development allowing the creation of systems customized to end-user culture, end-user role, and platform in use, as well as system re-use and evolution. It also supports consistency in interaction styles, particularly in web applications. This reasoning is made concrete through an example that presents a factory-automation prototype built using the BANCO architecture.

Categories and Subject Descriptors

D.1.7 [**Programming Techniques**]: Visual Programming; H.1.2 [**Models and Principles**]: User/Machine Systems – *Human Factors;* D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.6 [**Software Engineering**]: Programming Environments - *Interactive Environments*.

General Terms

Design, Human Factors, Languages.

Keywords

End-user development, unwitting software programming, interactive systems, AJAX architecture, XML.

1. INTRODUCTION

End users are "people outside the information system department", required "to develop software applications in support of organizational tasks" [5]. End-user development techniques propose various approaches that allow "users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend software artifacts" [18]. In this situation, end users are increasingly evolving from passive consumers of data and computer tools into active producers of information and software [8, 14]. Experts in industrial, business or scientific disciplines, not necessarily experts in computer science, end users are in any case responsible for possible errors and mistakes, even for those generated by wrong or inappropriate use of the software they develop. Also when given this responsibility, they do not willingly become computer experts, although they need to program and maintain control over the information-processing tasks they generate [7].

To overcome this contradiction, several researchers proposed approaches in which end users can program following their reasoning habits, and not computing habits of computer scientists, embedded in traditional programming languages [20, 23, 25]. In this line, this paper illustrates how EUD challenges can be faced by adopting the metaphor of habitable environments, first introduced by Alexander and Borchers seminal works [2, 4], which offer their users a "quality without a name" that allows them to develop their activities following strategies not prescribed a priori but dictated by the current situation. In the paper we present an architecture which implements the metaphor by presenting the virtual environments, offered to end users to develop their activities, as a space, in which virtual tools are deployed according to a presentation strategy reflecting end users' culture and expectations [6, 7]. These virtual tools are shaped and behave in ways that are familiar to end users. The virtual environment becomes a "space of opportunities", opportunities that end users can exploit to develop their reasoning and to achieve their goals. No predefined strategy is imposed on end users, who follow their reasoning strategies in solving problem - including when they develop new-programmed tools.

End users develop these programs as a part of their own activities, which they are highly motivated to perform but without being aware they are programming. For this reason and inspired by [21], they were called "unwitting programmers" or "unwitting software developers" in [9, 10], where their characteristics and the needs of appropriate development techniques are studied. Systems supporting unwitting user development and the tools populating it must not only be easy to use and easy to develop and to tailor but also designed in order to help users understanding and appropriating them [12]. To allow proper appropriation, the environments and the tools must not only be presented to end users as symbols (icons, words, graphical elements) which are familiar and easily perceptible to them, i.e. localized to the user culture [13], but must also be localized to the role the end user is acting and to the platform s/he is using, as discussed in the following sections.

This paper briefly introduces unwitting end-user development (uEUD) and then describes the current state of the specification of an architecture, which is designed to support uEUD. This architecture is the result of a process determined by the development of several interactive systems supporting different communities of unwitting software developers [6, 7, 8, 20].

This architecture is called BANCO architecture (B-architecture for short). It is an AJAX-oriented, open source architecture, designed a) to be customized to its user culture, to its user role and to the platform in use; b) to be easy to use and develop [11, 17]; c) to support end-user unwitting programming. At the base of the B-architecture there is a framework called BANCO framework supporting the unwitting development of interactive systems. This framework is composed by a set of XML and ECMAScript documents specifying the appearance and behavior of the system to be instantiated. The result of the instantiation activity is a customized version of the B-architecture, which appears to its users as an interactive Web environment in which they can develop their uEUD. The discussion is made concrete by the description of an example that presents a prototype realized for a factory automation company according to the B-architecture. In Section 2, we first introduce our view on uEUD. Next, the example is presented (Section 3). The B-architecture is described in Section 4. The specification documents in the BANCO framework and their corresponding languages are described in Section 5.

2. AN OVERVIEW OF UNWITTING END-USER DEVELOPMENT

The design and development of environments that support end users in unwitting software programming require knowledge spanning, at least, from software engineering to HCI (Human Computer Interaction) and specific application domain [3]. Therefore, at least software engineers, HCI experts and representative end users, as domain experts, must collaborate in the design. The whole process is not without difficulties. Members of the three communities, namely software engineers, HCI experts and end users, whether they are domain experts or workers on the field, must collaborate on the activity. Collaboration is made difficult by the communication gaps existing among them. For example, end users and software engineers actually possess distinct types of knowledge, users being the "owners" of the problem and developers being the "owners" of the technology to solve the problem. They follow different approaches and reasoning strategies for modeling, performing and documenting the tasks to be carried out in a given application domain; end users do not understand software developers' jargon and developers often do not understand user jargon. Interactive systems usually reflect the culture, skill and articulatory abilities of software engineers rather than those of end users. As a result, end users are obliged to adopt interaction styles that are alien to their culture, and are often charged with housekeeping tasks, which do not interest them and divert their attention from the activity they are performing [19]. Similar gaps arise between HCI experts and software engineers and HCI experts and end users [7, 16]. These clashes among cultures become particularly evident when the system requires end users to perform development activities. The problem is thus how to allow end users to define and develop their applications according to their own style of reasoning and to their mental models of the activities to be performed [19]. Furthermore, the achievement of complex activities by end users in their work environment requires the collaboration among end users of different cultures and experience. This will be illustrated in the example where shop foremen, line operators, and service engineers must collaborate in a production activity. End users from different communities also develop different user dialects, skills, knowledge, and notation [7]. Therefore, cultural clashes also characterize the activities and collaboration among end users working on the field. If, during system design, the clashes among different users' cultures are not taken into account, some users may be forced to adopt specific dialects related to the domain, but different from their own and possibly not fully understandable, thus making the collaboration difficult. Therefore, two contrasting requirements arise: (i) the need of guaranteeing an appropriate reasoning environment to every participant to the process; (ii) the need of guaranteeing an appropriate communication among the different participants. As to the first requirement, it is necessary that each participant is allowed to reason, experiment on prototypes and report her/his results using her/his modeling language and notations, which reflect her/his mental models; as to the second, it is necessary that different participants are able to communicate their results to each other and reach an adequate understanding of the reciprocal results. To fulfill these requirements we proposed an approach to uEUD which supports reasoning in the specific language of the cultural community the participant belongs to. To allow the proper appropriation [12] of the environment and its tools it is required to present to the users (both HCI experts, and end-users and software engineering experts) symbols (icons, words, sentences) which are familiar to them and easily perceptible that is environments customized according to the user culture, to the role of the user and to the platform in use. Working in these environments each participant develops and reasons on his model of the interaction process, works on prototypes and describes his insights and proposed improvements. To guarantee an efficient communication among these different participants each environment is equipped with an annotation tool by means of which the participant can make observations about the exchanged information and proposals about possible changes of the system in order to improve it. In this way, each member of the team can directly experiment with the system, see and critique the recommendations and notes of other team members and can negotiate the system evolution. Hence, the different workshops allow the creation and management of a shared knowledge base.

3. AN EXAMPLE OF UNWITTING DEVELOPMENT

We describe here an example of unwitting development, by illustrating a prototype we have developed for a factory automation company [8]. The company is in charge of developing the software and the user interfaces for the systems it sells to client companies. The company needs to create systems for factory automation that are usable for their client companies, that is easy to learn and to use, and that are customized for experts in factory automation but not in computer science. The company has also to develop software tools for supporting its personnel in the development, testing, and maintenance of the factory automation systems. The company personnel is organized into different categories of end users with different responsibilities and skills, which need to perform various tasks with the software tools. Since the client companies have different needs and habits, we design a virtual interactive system that can be shaped in different environments each one specific for a certain type of users.

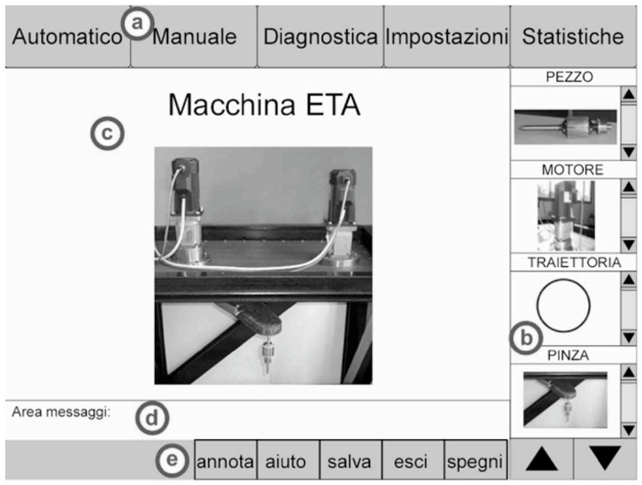


Figure 1. An application developed for a factory automation company. The letters have been added on the screenshot for the sake of explanation in the text.

In Figure 1 a prototype shaped for the assembly line operator, which is devoted to the control of a pick-and-place robot is shown. In this case, the assembly line operator needs to have (i) the possibility to choose among different robot's use modalities e.g. automatic, manual, diagnostics (see letter 'a' in Figure 1), (ii) the power to modify the robot's behavior or the task it has to perform by picking out the tools s/he needs and associate them to it (letter 'b' in Figure 1), and (iii) the possibility to access tools like annotation, online help, etc (letter 'e' in Figure 1). The assembly line operator can observe the behavior of the machine that is shown in the work area (c) and her/his activity is guided by the messages presented in the message area (d).

3.1 The Developed System in the Case of Factory Automation

As pointed out above, the structure of the company requires the design and development of different environments shaped for experts in factory automation, HCI experts, and end users working for the company. Therefore, we adopt the Software Shaping Workshop (SSW) methodology, already presented in [4], a meta-design participatory approach that does not end with the release of the software, but continues throughout the whole software life cycle. An interactive system is designed as a network of software environments called SSWs, or briefly workshops, each of them being either an environment through which end users perform their activities or an environment through which stakeholders participate in the design of the whole system, even at use time. An SSW is designed in analogy with an artisan or engineer workshop, namely a workroom where an expert finds all and only those tools necessary to carry out her/his activities. The SSW network of an interactive system is organized in three different levels based on the different types of activities the workshops are devoted to: the *VIS use level* includes workshops that are used by end users to perform their tasks (called application workshops); the *design level* includes workshops for designing and adapting the application workshops in accordance with the evolving knowledge and user needs (called system workshops); and the *meta-design level* includes the system workshop for software engineers, which allows them to generate and maintain all the workshops in the network [8]. In the case at hand, the environments at the design level are grouped into two levels (see Figure 2).

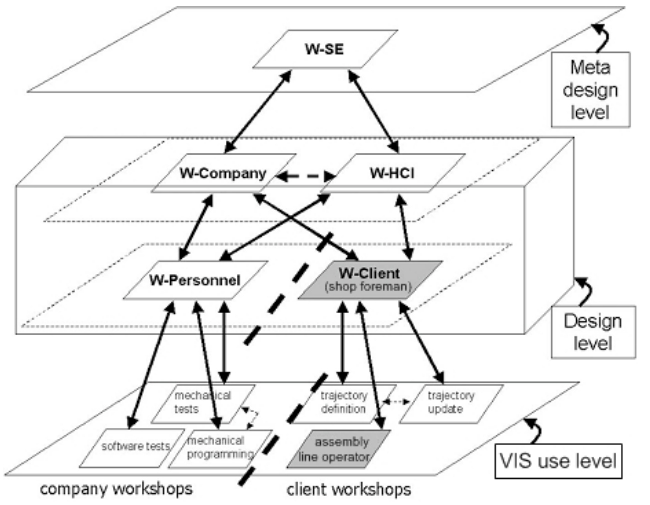
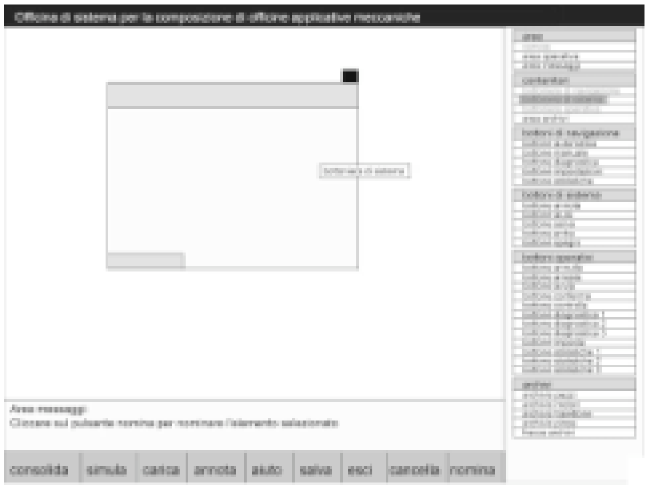


Figure 2. The environments for factory automation configured for the case discussed.

At the upper sublevel, the W-Company environment is devoted to the company experts who are in charge of creating all entities to be managed at the lower plane, while the W-HCI environment is used by HCI experts to check the created entities. At the bottom sublevel, the W-Personnel environment is used again by the company experts to generate the final environment, the W-Client, that will be used by the company end users. In our example the W-Client is used by the shop foreman.


Officina di sistema per la composizione di officine applicative meccaniche

Area messaggi:
Cliccare sul pulsante nomina per nominare l'elemento selezionato

Figure 3. Unwitting programming: the shop foreman drags and drops the entity "bottoniera di sistema" (system button panel) into the canvas representing the background of the line operator environment being created.

Two different types of activity can be identified, the first is the software mechanical design and testing of automation systems

and the second is the use of these automation systems in the client factory. The first activity is supported by the use of environments shaped for company's professionals, e.g. mechanical and software testers and mechanical programmers, while the second activity is performed by the use of environments devoted to client company's employee, e.g. production managers and assembly line operators (Figure 1).
Communications between different levels may occur from top levels to bottom levels and vice versa. Two types of documents can be exchanged: (i) programs and documents specifying the environments, for example the shop foreman creates and makes available an environment to the line operator; (ii) annotations about usability problems, new user needs or proposed improvements: for example, line operators can make available to the shop foreman requests for developing new tools to be used in their environments; whereas at the VIS use level, line operators can communicate to HCI experts their problems in understanding the meaning of certain data representations or how to use various tools.

3.2 A Bargain Between Shop Foreman and Line Operator

In this section, we describe a scenario in which the shop foreman, using W-Client, unwittingly programs the application workshop for the line operator through simple drag-and-drop activities (both environments are highlighted in Figure 2). The line operator, using her/his environment can communicate her/his difficulties. Figure 3 illustrates a snapshot of the W-Client at hand, during the interaction of a shop foreman for creating the line operator environment shown in Figure 1. In Figure 3, the shop foreman has already selected the entity "canvas" by choosing it from the menu on the right side: the "canvas" is the background of the environment that shop foreman is going to create. S/he has already added other entities and now is dragging and dropping the entity "bottoniera di sistema" (system button panel), which will appear on the "canvas" entity in its initial state. This is an example of unwitting programming. In Figure 4 the results of the shop foreman's activity on the line operator environment are presented.

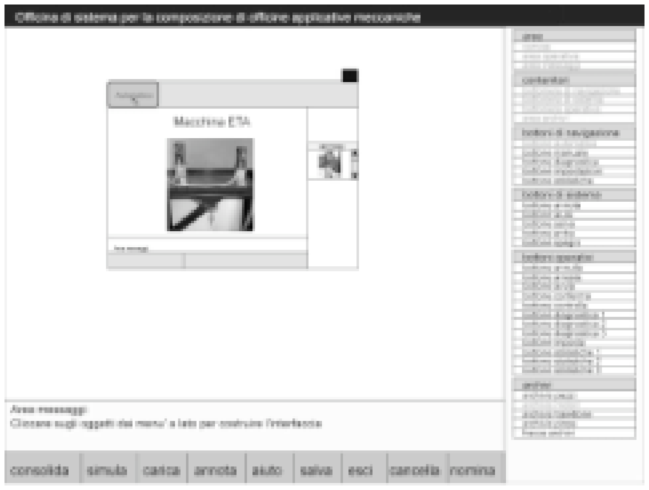

Officina di sistema per la composizione di officine applicative meccaniche
Macchina ETA
Area messaggi:
Cliccare sugli oggetti dei menu' a lato per costruire l'interfaccia
consolida
simula
carica
annota
aiuto
salva
esci
cancella
nomina

Figure 4. Unwitting programming: the shop foreman is creating the line operator environment shown in Figure 1. The button "automatico" (automatic) is located on the operative button panel.

S/he is now positioning a new button on the operative button panel. Once the shop foreman considers the realization of the line operator environment completed, s/he releases it using the "salva officina" functionality. This functionality creates an instance of the line operator environment and makes it available to her/him. The line operator accesses the new environment and starts working with it. In performing her/his activity s/he finds usability problems: s/he does not understand how to perform the robot's diagnostics because the provided strategy is different from the one s/he traditionally adopts in her/his application domain. Therefore, s/he chooses the "annotation mode" selecting the "annota" button from the "tools" menu. Then s/he selects the "Diagnostica" button. A buttons panel appears, providing tools that permit the insertion of textual annotations.

The system opens the annotation form and s/he annotates her/his observations. Figure 5 shows a screenshot taken during the annotation of the line operator environment performed by the line operator. The shop foreman receives the observations and discusses them with all the other members of the design team to satisfy the end-user requests. Therefore, the design activity is developed as a bargaining activity among the different end users and designers, aimed at balancing requirements on the system that arise from the different points of view.

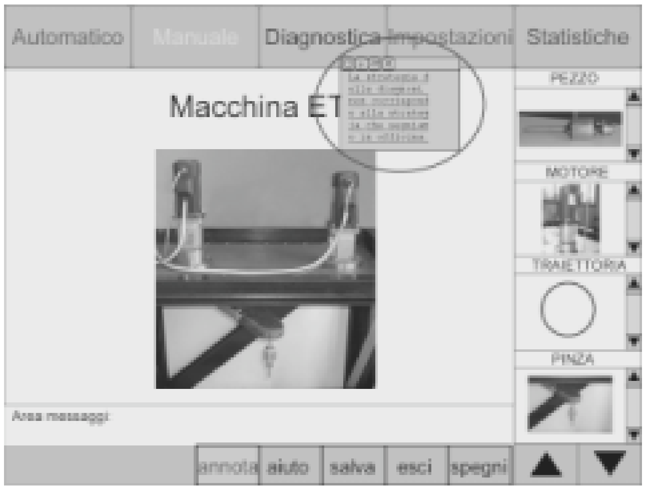


Figure 5. The line operator annotates the assembly line environment to communicate her/his problems to the designers. The circle has been added on the screenshot for the sake of explanation in the text.

4. BANCO ARCHITECTURE

The B-architecture emerges from an evolutionary process of experiences based on unwitting programming and participatory design [24]. The novelties of this architecture are related to the need of the following uEUD requests: (i) system customization, in that users need to access systems customized to their culture, to their roles and to the platforms they use; (ii) system evolution, in that users need to evolve their systems according to their evolving needs and habits; (iii) communication, in that users need to communicate with each other exchanging not only documents but also prototypes that materialize their experiences. In the architecture presented in Figure 6, on the server side, there are two archives: (i) the archive of the BANCO framework elements; and (ii) the archive of the knowledge base of the current application. The BANCO framework elements archive contains the documents that specify the BANCO frameworks. This archive is accessible by the whole community of BANCO users and developers. The knowledge base of the current application archive contains the working documents generated in the design and development of the environments of a specific application. This archive is accessible by the community of the developers of the specific application. Designers can store a copy of a working document as a BANCO framework, making it available to the whole community of BANCO implementers. In the example, the shop foreman is initially developing the line operator environment, as shown in Figure 3 and 4. These documents, which specify the line operator environment, are executable (by clicking on the "Simula" button). During this activity, the shop foreman can store these documents in the knowledge base of the current automation application. The documents become available to every designer in the design team. At the end of this activity, the shop foreman releases the line operator environment by selecting the salva officina button. As a result, a copy of the set of documents s/he has defined is stored in the BANCO framework archive and becomes available to the line operator and to the BANCO community developers.

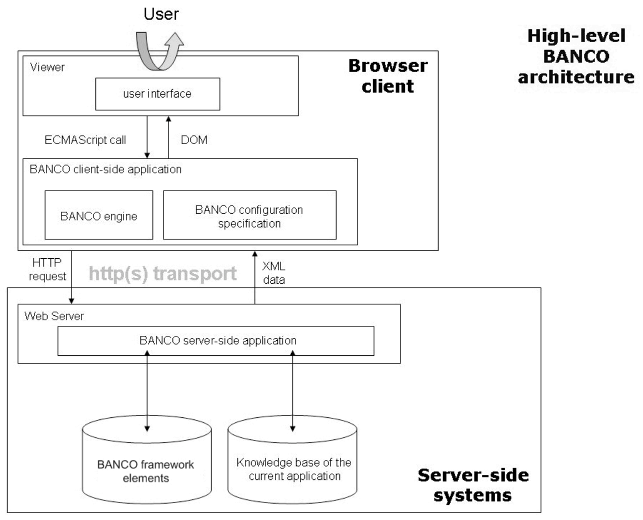

User
High-level BANCO architecture
Viewer
Browser client
user interface
ECMAScript call
DOM
BANCO client-side application
BANCO engine
BANCO configuration specification
HTTP request
http(s) transport
XML data
Web Server
BANCO server-side application
BANCO framework elements
Knowledge base of the current application
Server-side systems

Figure 6. A high-level view of the B-architecture.

The BANCO framework elements archive represents the blackboard for the design team members. The Knowledge base of the current application archive represents the blackboard for the users that are developing their own environment. The B-architecture comprises on the server side a BANCO server-side application, which manages the retrieve of the BANCO framework elements and the retrieve and storage of the working documents stored in the Knowledge base of the current application archive. On the client side, an instance of the BANCO engine, currently an ECMAScript program, interprets the configuration documents, i.e., the BANCO framework elements and the working documents retrieved on the server side. It creates the static part of the system, using a library of instantiation functions, and manages the dynamic of the system (system's reactions to user's actions), by using a library of interaction management functions. The configuration documents specify content, organization, localization, materialization and interaction dynamics of the current configuration. The presence of a Web browser equipped with a Viewer allows the dynamical creation of the current configuration and the management of the I/O interaction between the user and the system by communicating with the BANCO engine in different steps of the session.

4.1 The Tools for BANCO Customization

Current BANCO elements' implementations are based on different languages, which permit the customization of the different environments. In fact, the ability of a system to be customized to the user's culture, her/his role, and the platform in use requires four level specifications: 1) content and organization, 2) localization, 3) materialization components and 4) interaction dynamic level specifications. These requests lead us to the definition of two XML-based specialized languages (IM^{2}L and LML), one SVG-based or HTML-based language, the Template Language, and to the use of ECMAScript to steer the dynamics of the interaction.

IM^{2}L (Interaction Multimodal Markup Language) is an XML-based markup language whose elements are documents that describe a virtual environment as a "space of opportunities" abstracting from users' culture, their roles, and the platforms they use to access the environments.

IM^{2}L is defined by a XML Schema called IM^{2}L schema. Each node in the schema defines a type of virtual entity (ve), which may appear in an IM^{2}L system. A ve is a dynamic system that captures the user inputs, compute the reaction to them and materialize its own state - the results of the computation – in a form perceivable by the user [15].

For example, a type of ve "operator" indicates a component associated to the execution of a computational activity without the need to specify the materialization properties related to this component. At the materialization time, this ve "operator" will be instantiated, for example, in a button or in or a menu item according to the system context of use. An "operator" belongs (is superimposed) to an "operatorSet" (for example, a menu item belongs to a menu, a button belongs to a button panel).

Each node of the IM^{2}L schema has the following format:

1. name
2. the number of states in its dynamics
3. the list of the names of the admissible states of the ve
4. the initial state that the ve has as it is instantiated
5. the next states and output functions for each admissible state of the ve
6. the procedures associated to each state (the initial state is generally associated to the "null" procedure)
7. the emotion associated to each state (for example, a light alarm can be in two states: normal, which communicates a relaxing emotion, and alarm, which transmits an alert emotion)
8. the ve to which it is superimposed
9. order and topological attributes in the ve to which it is superimposed
10. the ves which can be superimposed to it

The IM^{2}L schema is used to define a specific system, i.e. an environment dedicated to one application.

Through the unwitting programming technique, end users have the possibility to design proper applications for specific domains. In our approach, the end user during design exploits two different XML documents that are able to specify the context of the application domain: a workshop schema (W-schema) and a workshop garden centre (W-garden).

The W-schema describes a subset of the IM^{2}L schema reporting only the ve types suitable for a specific domain. For example, the "operator" node in the W-schema of the shop foreman workshop indicates that, in this domain, the possible operators are only: menu items, buttons and sliders.

The W-schema is used to define a set of IM^{2}L documents which constitute a W-garden. Each document in the W-garden describes the initial state of a ve which can be used in the application. The W-garden is stored in the BANCO Framework Elements archive on the server. When required by the interaction process, the client retrieves the document from the server and feeds it to the BANCO Engine. The BANCO Engine generates the corresponding ve by interpreting the document which describes the initial state of the ve and builds a DOM tree which is joined to the DOM tree describing the current state of the whole system: this action recalls the grafting of a branch on a tree.

Back to the example, let us examine the IM^{2}L definition of the "salva officina" button, appearing on the left bottom corner of Figure 3 and 4. When the shop foreman selects the button, the procedure to save the current defined line operator environment which appears in the work space is stored as a framework and made available to designers and users. Its IM^{2}L definition states that: an active element exists, it has a name – *store framework activation*-two states – *selected, non selected-*, its initial state is *non-selected*, its selection fires the *save-framework* procedure: moreover the active element is a normal element- that is it not an alarm or a "requiring special attention" button; it is the first of an ordered set of active elements – *shop foreman actions-* , to be represented in the same container – *shop foreman action container*.

The end-user using the W-schema and W-garden is able to design final applications in an abstract way that is the graphical attributes (dimension, colors, shape…) of the ve are not instantiated. The Template Language (TL) specifies how the ve states must be materialized (i.e. represented on the screen or emitted if sound, etc.). The elements of TL are SVG (Scalable Vector Graphics) or HTML documents, which specify shape, relocatable geometry, and graphical properties (such as the lines thickness) of the materialization of each state of each ve whose initial state is described in W-garden. These documents have a common format described by a TL schema.

In the salva officina functionality, TL states that rectangular shapes are formed by four orthogonal, linked segments, that the salva officina button is formed by segments of given thickness, length: also defines black, light and dark gray RGB coordinates. Moreover, TL states that Italian texts are in Arial, of given size and style.

LML (Localization Markup Language) is an XML-based markup language for the specification of the localization properties of an interactive system, i.e., of those properties that in an interactive IM^{2}L-based system characterize geometry, topology, colour representation, shapes and text, depending on the culture of the user, the user's role and the platform s/he uses as well as user physical capabilities. LML is specified by the LML schema in which for each ve in W-schema the localization properties are defined. The properties currently taken into account are shape, colour, orientation, size, position, order, and thickness.

For the salva officina functionality, its LML definition states that the element *store framework activation* is defined according to the Italian culture, mechanical technical standards, and company style guideline. The Italian culture states that the name will be in Italian language, that the order of representation of active elements in a button panel is from left to right; and that salva officina being the first must be on the left of the action container. Technical standards suggest that the name is "*salva officina*" and that the colours of normal active element state representations should be relaxing. Company style guidelines state that such an active element will be a *button*, that active elements like buttons are represented as *rectangular shape* of *black* colour; that ordered sets of active elements are organized in *button panels*, and that salva officina must be coloured in *light gray* in the non-selected state and *dark gray* in the selected state.

4.2 Use of the Tools for BANCO customization

When a user accesses the workshop, by clicking on its icon, the Bowser loads BANCO engine from the archive of BANCO frameworks. BANCO engine starts the instantiation of a system or an application workshop by:

- checking the user locale
- loading IMML specification of the workshop of the application from archive of the BANCO framework
- loading LML specification of the workshop fitting the user profile from archive of BANCO framework
- loading current templates of the workshop associated to the application from archive of BANCO framework
- localizing the templates according to LML document
- making the initial state of the workshop available to the user

BANCO engine creates a new ve, when required by the interaction process at a certain time t, by:

- loading IMML specification of the ve from archive of BANCO framework
- using available templates and LML document to localize the templates pertaining to the ve at hand
- creating a ve - DOM tree which specifies the ve at hand
- joining the ve - DOM tree to the application A-DOM tree describing the state of the application at time t
- materializing the updated A- DOM tree by externalizing the application state at t+1

These languages and libraries are used to create the framework elements. A BANCO framework is composed of the following documents:

- The HTML access page: it allows the users to identify themselves. On the base of the users' identification, provides data to retrieve the user's profile, the role profile, and the platform profile.
- The XML profile document: describes the user activity, the user role and a link to the associated localization document.
- The Starter document: it specifies user, user's role, and platform. The Starter document is interpreted by the browser's viewer, which retrieves the set of configuration documents in the BANCO framework elements archive and

in the Knowledge base archive on the system server side, according to the profiles specified in the Starter. The Starter document is written in the language chosen for the system materialization (e.g. SVG or HTML) based on the users' preferences, the device in use (e.g. desktop PC or PDA) and the Web browser used for accessing the system.

- System Initial State Specification is the IM^{2}L document that specifies the static part of content and organization of the system initial state;
- Components Initial State Specification is a set of IM^{2}L documents, each one specifying a type of entity that can be instantiated during the interaction process to modify the state of the system;
- Localization Documents, written in LML language, specifying the localization properties of the system, in particular geometry, topology, colours, shape and text.
- Templates are SVG-based or HTML-based documents, each one describing the physical materialization of a type of entity composing the system.
- BANCO engine's instantiation library: set of functions that manage the materialization of the initial state of the interactive environment.
- BANCO engine's interaction management library: set of functions that manage the interaction between the user and the interactive environment.

If the Starter document is an HTML document, the HTML Templates are used and the whole system will be an HTML system. If the Starter document is an SVG document, the SVG Templates are selected and the whole system will be an SVG system. In this case, the system needs an SVG viewer in the template to be interpreted. Unlike HTML, SVG allows a) the management of the screen at pixel level, i.e., annotations can be associated to every segment of an image; b) the management of the shapes. Therefore, an HTML version of a system, defined by the same IM^{2}L and LML documents, is degraded with respect to its SVG counterpart.

In the current implementation, we adopt as SVG Viewer the Adobe SVG Viewer Plug-in [1]. We also chose to manage both the archive of the BANCO framework elements and the one of the Knowledge Base of the current application with eXist, an open source native XML database. The BANCO server-side application is written using PHP scripting language [22].

5. CONCLUSION

This paper introduces unwitting end-user development and the BANCO architecture, which is designed to support unwitting end-user development. An interactive system implemented according to the BANCO architecture is localized to its user culture, to the user's role, and to the platform in use. To reach this goal, three XML languages, which specify an interactive system at different levels of abstraction, are introduced. IM^{2}L describes the – abstract and internationalized – content of an interactive system, i.e., its functional components as well as its logical structure. LML describes the localization properties (e.g. geometry, topology, colour representation) that depend on end user culture, on end user role and on platform in use as well as their physical capabilities. Finally, TL describes how to materialize the interactive system customized for a given end user. A set of documents written in these languages, the BANCO framework, is interpreted by the BANCO engine to create and manage an interactive environment, a space of opportunity, customized to end-user culture, end-user role, and platform in use. The localization of the environments and of the tools they offer to users' habits allows the appropriation process. Not all the stakeholders involved in the process of developing and maintaining application workshops for uEUD are unwitting programmers (Figure 2). Software engineers are experts in programming and often HCI experts and some domain experts may also have some programming experience. The BANCO architecture allows to support also their work permitting the instantiation of interactive environments that make their programming activities easier and make easier the development of consistent application workshops.

6. ACKNOWLEDGMENTS

The authors would like to thank the ETA Consulting in Brescia (and particularly Ing. Silvano Biazzi) for their collaboration in the case study and the reviewers for their useful comments. The present work is partially funded by the 12-1-5244001-25009 FIRST grant of the University of Milan.

7. REFERENCES

[1] Adobe SVG viewer 2008. http://www.adobe.com/svg/.

[2] Alexander, C. 1977. A Pattern Language: Towns, Buildings, Construction. Oxford University Press.

[3] Arias, E., Eden, H., Fischer, G., Gorman, A., and Scharff, E. 2000. Transcending the individual human mind—creating shared understanding through collaborative design. ACM Transactions on Computer-Human Interaction (TOCHI), 7, 1, (2000), 84-113.

[4] Borchers, J. 2001. A Pattern Approach to Interaction Design. Wiley.

[5] Brancheau, J. C., and Brown, C. V. 1993. The Management of End-User Computing: Status and Direction. ACM Computing Surveys. 25, 4 (1993), 437-482.

[6] Carrara, P., Fresta, G., and Rampini, A. 2000. Interfaces for geographic applications on the World Wide Web: an adaptive computational hypermedia. In Proceedings of the 6th ERCIM Workshop on "User interfaces for all" (Florence, Italy, October 25-26, 2000). 341-342.

[7] Costabile, M. F., Fogli, D., Mussio, P., and Piccinno, A. 2006. End-user Development: The Software Shaping Workshop Approach. In End-User Development, H. Lieberman, F. Paternò and V. Wulf, Eds. Springer, Dodrecht, The Netherlands, 1-8.

[8] Costabile, M. F., Fogli, D., Mussio, P., and Piccinno, A. 2007. Visual interactive systems for end-user development: A model based design methodology. IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans. 37, 6 (2007), 1029-1046.

[9] Costabile, M. F., Mussio, P., Parasiliti Provenza, L., and Piccinno, A. 2008. Advanced Visual Systems Supporting Unwitting EUD. In Proceedings of the Working Conference on Advanced Visual interfaces (Napoli, Italy, May 28 - 30, 2008). AVI '08. ACM, New York, NY, 313-316.

[10] Costabile, M. F., Mussio, P., Parasiliti Provenza, L., and Piccinno, A. 2008. End users as unwitting software developers. In Proceedings of the 4th International Workshop on End-User Software Engineering (Leipzig, Germany, May 12, 2008). WEUSE '08. ACM, New York, NY, 6-10.

[11] Crane, D., Pascarello, E., and James, D. 2005. AJAX in action. Manning Publishing Company.

[12] Dix, A. 2007. Designing for Appropriation. In Proceedings of the XXI Conference on People and Computers (Lancaster, UK, September 3-7, 2007). HCI 2007. 7-10.

[13] Esselink, B. 2000. A practical guide to localization. John Benjamins.

[14] Fischer, G. 2002. Beyond 'Couch Potatoes': From Consumers to Designers and Active Contributors. FirstMonday (Peer-Reviewed Journal on the Internet), Available at http://firstmonday.org/issues/issue7_12/fischer/.

[15] Fogli, D., Fresta, G., Marcante, A., and Mussio, P. 2004. IM2L: a user interface description language supporting electronic annotation. Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages (Gallipoli, Italy, 2004) AVI 04.

[16] Folmer E., van Welie M., and Bosch J. 2005. Bridging patterns: An approach to bridge gaps between SE and HCI. Journal of Information and Software Technology. 48, 2 (2005), 69-89.

[17] Garrett, J. J. 2005. Ajax: A New Approach to Web Applications. http://www.adaptivepath.com/ideas/essays/archives/000385.php

[18] Lieberman, H., Paternò, F., Klann, M., and Wulf, V. 2006. End-User Development: An Emerging Paradigm. In End-User Development, H. Lieberman, F. Paternò and V. Wulf, Eds. Springer, Dodrecht, The Netherlands, 1-8.

[19] Majhew, D.J. 1992. Principles and Guideline in Software User Interface Design. Prentice Hall.

[20] Mussio, P., Pietrogrande, M., and Protti, M. 1991. Simulation of hepatological models: a study in visual interactive exploration of scientific problems. JVLC. 2, 1 (1991), 75-95.

[21] Petre, M., and Blackwell, A. F. 2007. Children as Unwitting End-User Programmers. In Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (Coeur d'Alène, Idaho, USA, September 23-27, 2007). VL/HCC 2007. 239-242.

[22] PHP 2008. http://www.php.net/

[23] Repenning, A., Ioannidou, A. 2006. What makes End-user development tick? 13 Design Guidelines. In End-User Development, H. Lieberman, F. Paternò and V. Wulf, Eds. Springer, Dodrecht, The Netherlands, 51-85.

[24] Schuler, D., and Namioka A. (eds.) 1993. Participatory Design - Principles and Practices. Lawrence Erlbaum Associates.

[25] Whitley, K. N., and Blackwell, A. F. 2001. Visual Programming in the Wild: A Survey of LabVIEW Programmers. JVLC. 12, 4 (Aug. 2001), 435-472.