PsychNology Journal, 2004
Volume 2, Number 1, 99 - 122

# Software Environments for End-User Development and Tailoring

Maria Francesca Costabile♥[1], Daniela Fogli[2],
Giuseppe Fresta[3], Piero Mussio[4], Antonio Piccinno[1]

[1] Dipartimento di Informatica, Università di Bari, Bari, Italy.

[2] Dipartimento di Elettronica per l'Automazione, Università di Brescia, Brescia, Italy
[3] ISTI - CNR, Pisa, Italy.

[4]Dipartimento di Informatica e Comunicazione, Università di Milano, Milano, Italy.

**ABSTRACT**

In the Information Society, end-users keep increasing very fast in number, as well as in their demand with respect to the activities they would like to perform with computer environments, without being obliged to become computer specialists. There is a strong request of providing end-users with powerful and flexible environments, tailorable to the culture, skills and needs of very diverse end-user population. In this paper, we discuss a framework for End-User Development (EUD) and present our methodology to design software environments that support the activities of a particular class of end-users, called domain-expert users, with the objective of easing the way these users work with computers. Such environments are called Software Shaping Workshops in analogy to artisan workshops, since they provide users with the tools, organized on a bench, that are necessary to accomplish their specific activities by properly shaping software artifacts. The methodology is discussed, outlining its implementation through a web-based prototype.

Keywords**:** *End-User Development, Visual Interaction, Tailoring, Customization*.

Received 16 January 2004; received in revised form 31 March 2004; accepted 1 April 2004.

## 1. Introduction

Even if progress has been made in improving the way users can access interactive software systems, some phenomena affecting the life of interactive products still make difficult to develop software systems acceptable in a working environment. In Human-Computer Interaction (HCI), it is often observed that "using the system changes the users, and as they change they will use the system in new ways" (Nielsen, 1994). In turn, the system must evolve to adapt to its new usages; we called this phenomenon

♥ Corresponding Author:
Maria Francesca Costabile
Dipartimento di Informatica,
Università di Bari, Bari, Italy
E-mail: costabile@di.uniba.it

*co-evolution of users and systems* (Arondi et al., 2002). In (Bourguin et al., 2001), it is observed that these new uses of the system determine the evolution of the users culture and of their models and procedures of task evolution, while the requests from users force the evolution of the whole technology supporting interaction.

Co-evolution stems from two main sources: a) user creativity, i.e. users may devise novel ways to exploit the system in order to satisfy some needs not considered in the specification and design phase; and b) user acquired habits, i.e. users may follow some interaction strategy to which they are (or become) accustomed; this strategy must be facilitated with respect to the initial design.

Co-evolution implies tailoring that, according to (Mørch and Mehandjiev, 2000), is "the activity of modifying an existing computer system in the context of its use, rather than in the development context". This definition emphasizes that users themselves can tailor the system to their necessities. Tailoring stems from a continuous adaptation of a system and is seen as the indirect long-term collaboration between developers and users. Tailoring should be driven by users to exploit the potential benefits of task-oriented and skill-based system adaptations that only users themselves can perform. However, a trade-off to this approach is the variety of developed applications to be maintained by software engineers. Our proposal is also aimed at coping with this problem.

One fundamental challenge for the coming years is to develop environments that allow people without particular background in programming to develop and tailor their own applications, still maintaining the congruence within the different evolved instances of the system. Over the next few years, we will be moving from *easy-to-use* (which has yet to be completely achieved) to *easy-to-develop-and-tailor* interactive software systems.

People who will mainly benefit from this perspective shift are *end-users*, i.e. those persons who, according to Cypher, use a computer application as part of daily life or daily work, but are not interested in computers per se (Cypher, 1993). It is evident that several categories of end-users can be defined, for instance depending on whether the computer system is used for work, for personal use, for pleasure, for overcoming possible disabilities, etc. End-user population is not uniform, but divided in non-mutually exclusive communities characterized by different goals, tasks and activities. Even these communities cannot be considered uniform, because they include people with different cultural, educational, training, and employment background, novice or experienced in the use of the computer, the very young and the elderly, and with

different types of (dis)abilities. End-users operate in various interaction contexts and scenarios of use, they want to exploit computer systems to improve their work, but often complain about the difficulties in the use of such systems. The challenge for designers is to develop interactive systems customized to a community, without loosing the generality and the power of computer tools.

Our experience is focused on a particular class of end-users, that we call *domain-expert users* (or *d-experts* for short): this kind of users, such as medical doctors, mechanical engineers, geologists, etc., are experts in a specific domain, not necessarily experts in computer science, and use computer environments to perform their daily tasks. D-experts have the responsibility for possible errors and mistakes, even those generated by wrong or inappropriate use of the software.

This paper describes a method to overcome the trade-off between customization and generality in the case of d-experts.

The ultimate aim is empowering end-users and d-experts to flexibly employ advanced information and communication technologies within the future environments of ambient intelligence. To this aim, the European Community has recently funded EUD-Net, a network of Excellence on End-User Development (EUD).

This paper provides the following contributions to the research on EUD:

1) an analysis of the need of developing software that domain-expert users have;
2) a design methodology to build
   a. software environments tailored to the needs of the considered user community,
   b. tools allowing d-experts to design and develop software environments in collaboration with software engineers and HCI experts.

The proposed design methodology is the evolution of the design strategy described in (Carrara et al., 2002a, Costabile et al. 2002), and allows both end-users and d-experts to perform EUD activities, including tailoring of existing software environments and creation of new software artifacts.

The paper is organized as follows: Section 2 provides insights to the concept of EUD. In Section 3, an analysis of needs of EUD, that domain-expert users have, is reported. Section 4 illustrates the Software Shaping Workshop methodology. Section 5 discusses the Interaction Visual Language used in the design methodology. Section 6

presents an example of the application of the SSW methodology to a real case. Section 7 concludes the paper.

## 2. End-User Development

New technologies have created the potential to overcome the traditional separation between end-users and software developers. New environments able to seamlessly move from using software to programming (or tailoring) can be designed. Advanced techniques for developing applications can be used by individuals as well as by groups or social communities or organizations.

Some studies say that by 2005, there will be in USA 55 millions of end-users compared to 2.75 millions of professional users (Boehm et al., 2000). End-users population is not uniform, but it includes people with different cultural, educational, training, and employment background, novice and experienced computer users, the very young and the elderly, people with different types of (dis)abilities. Moreover, these users operate in various interaction contexts and scenarios of use and they want to exploit computer systems to improve their work, but often complain about the difficulties in the use of such systems.

Based on the activity performed so far within the EUD-Net network of excellence, the following definition of EUD has been proposed: "*End-User Development is a set of activities or techniques that allow people, who are non-professional software developers, at some point to create or modify a software artifact".* EUD means the active participation of end-users in the software development process. In this perspective, tasks that are traditionally performed by professional software developers are transferred to the users, who need to be specifically supported in performing these tasks. The range of active user participation in the software development process can range from providing information about requirements, use cases and tasks, including participatory design, to end-user programming. Some EUD-oriented techniques have already been adopted by software for the mass market such as the adaptive menus in MS Word™ or some programming-by-example techniques in MS Excel™. However, we are still quite far from their systematic adoption.

EUD is based on the differences among end-users, professional programmers and software engineers. There are differences in training, culture, skill and technical abilities, in the scale of problems to be solved, in the processes, etc.

Within the EUD-Net activity, the following research directions have been identified as fertile for allowing end-users to *craft* software: 1. theoretical and empirical studies of what problems addressed by software engineering transpose to EUD, why and how; 2. studies to identify possibly existing problems that are specific to EUD and are thus not addressed by software engineering; 3. research on methods and tools that would address the previously identified problems in ways that are adequate for end-users: "lightweight methods", tools to support them, and offering appropriate user interfaces taking into account end-users tasks and activities.

Our proposal of designing Visual Interactive Systems organized as environments called Software Shaping Workshops, which will be illustrated in Section 4, is in the direction of point 3 above.

## 3. User needs of EUD

We often work with end-users that are experts in their field, that need to use computer systems for performing their work tasks, but that are not and do not want to become computer scientists. This has motivated our definition of domain-expert users.

In our work, we primarily address the needs of communities of d-experts in scientific and technological disciplines. These communities are characterized by different technical methods, languages, goals, tasks, ways of thinking, and documentation styles (Varela, 1979). The members of a community communicate among them through documents, expressed in some notations, which represent (materialize) abstract or concrete concepts, prescriptions, and results of activities. Often, dialects arise in a community, because the notation is applied in different practical situations and environments. For example, technical mechanical drawings are organized according to standard rules which are different in Europe and in USA (ISO 5456). Explicative annotations are written in different national languages. Often the whole document (drawing and text) is organized according to guidelines developed in each single company. The correct and complete understanding of a technical drawing depends on the recognition of the original standard as well as on the understanding of the national (and also company developed) dialects.

Recognizing users as d-experts means recognizing the importance of their notations and dialects as reasoning and communication tools. It also suggests to develop tools customized to a single community. Supporting co-evolution requires in turn that the tools developed for a community can be tailored by its members to the newly emerging

requirements (Mørch and Mehandjiev, 2000). Tailoring can be performed only after the system has been released and therefore when it is used in the working context. In fact, a contrast often emerges between the user working activity, which is situated, collaborative and changing, and the formal theories and models that underlie and constitute the software system. This contrast can be overcome by allowing users to adapt themselves the system they are using.

Recognizing the diversity of users calls for the ability to represent a meaning of a concept with different materialization, e.g. text or image or sound, and to associate to a same materialization a different meaning according, for example, to the context of interaction. For instance, a same interface of a distributed system in the automation field, is interpreted in different ways by a technician and a worker. These two d-experts are however collaborating to get a common goal. For this, they use a same set of data, which is however represented according to their specific skills. This is a common case: often experts work in a team to perform a common task. The team might be composed by members of different sub-communities, each sub-community with different expertise. Members of a sub-community should need an appropriate computer environment, suitable to them to manage their own view of the activity to be performed.

When working with a software application, d-experts feel the need to perform various activities that may even lead to the creation or modification of software artifacts, in order to get a better support to their specific tasks, thus being considered activities of EUD. The need of EUD is a consequence of user diversity and user evolution we have discussed. Moreover, the interactive capabilities of new devices have created the potential to overcome the traditional separation between end-users and software developers. New environments able to seamlessly move between using and programming (or customizing) can be designed.

Within EUD, we may include various tailoring activities. Indeed, tailoring activities are defined in different ways in the literature; they include adaptation, customization, end-user modification, extension, personalization, etc. These definitions partly overlap with respect to the phenomena they refer to, while often the same concepts are used to refer to different phenomena. In (Wulf, 1999), tailorability is defined as the possibility of changing aspects of an application's functionality during the use of an application, in a persistent way, by means of tailored artefacts; the changes may be performed by users that are local experts. Tailorability is very much related to adaptability. Different meanings are associated to tailorability and adaptability. To avoid ambiguity, two classes of d-expert activities have been proposed in (Costabile et al., 2003):

*Class 1*. It includes activities that allow users, by setting some parameters, to choose among alternative behaviors (or presentations or interaction mechanisms) already available in the application; such activities are usually called parameterization or customization or personalization.

*Class 2*. It includes all activities that imply some programming in any programming paradigm, thus creating or modifying a software artifact. Since we want to be as close as possible to the human, we will usually consider novel programming paradigms, such as programming by demonstration, programming with examples, visual programming, macro generation.

While many systems exist which support the performance of activities of class 1, our EUD methodology aims at the development of systems which allow d-experts to perform activities of class 2, as we will see in the following sections.

## 4. Software Shaping Workshops

The methodology we have developed to design visual interactive systems considers the following features: 1) adopting the user notation in the system development; 2) offering different views of the activity to the various members of the same community; 3) allowing end-users to participate to system tailoring; 4) guaranteeing a gentle slope of complexity (Myers et al., 2003). The latter means that, in order to be acceptable by its users, the system should avoid big steps in complexity and keep a reasonable trade-off between ease-of-use and expressiveness. Systems might offer for example different levels of complexities, going from simply setting parameters to integrating existing components, up to extending the system by programming new components. To feel comfortable, users should work at any time with a system suitable to their specific needs, knowledge, and task to perform. To keep the system easy to learn and easy to work with, only a limited number of functionalities should be available at a certain time to the users, those that they really need and are able to understand and use. The system should then evolve with the users, thus offering them new functionalities only when needed.

More precisely, our approach to the design of visual interactive systems for specific communities of d-experts is to organize a system as composed of various environments, each one devoted to a specific sub-community. Such environments are organized in analogy with the artisans workshops, where the artisans find all and only the tools necessary to carry out their activities. Following the analogy, d-experts using a virtual workshop find available all and only the tools required to develop their

activities. These environments are called *application workshops*, because they allow d-experts to perform their daily tasks.

Using an application workshop, d-experts of a sub-community can work out data from a common knowledge base and produce new knowledge, which can be added to the common knowledge base. All the data available for the community are accessible by each d-expert using the specialist notation of its sub-community.

The application workshops are designed by a design team composed by various experts, who participate to the design using workshops tailored to them. These workshops are called *system workshops* and are characterized by the fact that they are used to generate or update other workshops. Using a system workshop, some experts of the design team defines notations and tools, which are added to the common knowledge base and made available in the generated workshops.

This approach leads to a workshop hierarchy that tries to bridge the communicational gap between software engineers and experts of the application domain, since all cooperate in developing computer systems customized to the needs of the users communities without requiring them to become skilled programmers.

The system workshop at the top of the hierarchy is the one used by the software engineers. Each system workshop is exploited to incrementally translate concepts and tools expressed in computer-oriented languages into tools expressed in notations that resemble the traditional user notations, and therefore understandable and manageable by users. More precisely, at each level of the hierarchy but the bottom level, people use a system workshop and might create a child workshop tailored to a different type of d-expert.

The hierarchy organization depends on the working organization of the user community to which the hierarchy is dedicated: each hierarchy is therefore organized into a number of levels. The top level (software engineering level) and the bottom level (application level) are always present in a hierarchy. The number of intermediate levels is variable according to the different working organization of the user community to which the hierarchy is dedicated (Carrara et al., 2002b) and to guarantee a gentle slope of complexity.

Both application and system workshops are *Software Shaping Workshops* (SSWs): interacting with them, users get the feeling of simply manipulating the objects of interest in a way similar to what they might do in the real world. They are 'shaping' software, in that: a) by using a system workshop, d-experts actually create a software artifact, without writing any textual program code, but using high level visual languages

tailored to their needs; b) using an application workshop d-experts adapt the appearance and behavior of the available tools according to their culture, skills, and background.

To make clear the above concepts, in Section 6 we refer to a prototype under study in the system automation field, designed to support different communities of workers and technicians.

## 5. A view on visual interaction

To develop a Visual Interactive System organized as SSW hierarchy, software engineers and d-experts have first to define the pictorial and semantic aspects of the Interaction Visual Languages through which users interact with workshops. In our approach, we capitalize on the theory of visual sentences developed by the Pictorial Computing Laboratory (PCL) and on the model of WIMP (Windows, Icons, Menus, Pointers) interaction it entails (Bottoni et al., 1999). From this theory, we derive the formal tools to obtain the definition of Interaction Visual Languages.

The PCL model recognizes the interaction between users and interactive systems as a syndetic process in which systems of different nature (the cognitive human - the 'mechanical' machine) cooperate to achieve a task (Barnard et al., 2000). The different systems interact by communicating, interpreting and materializing sequences of messages at successive instants of time. If we restrict to the case of WIMP interaction (Dix et al., 1998), the messages exchanged are the whole images which appear on the screen display of a computer and are formed by text, icons, graphs, pictures, windows. Two interpretations of each element on the screen and of each action arise during the interaction: one performed by the user, depending on his/her role in the task, as well as on his/her culture, experience, and skills, and the second internal to the system, associating the image with a computational meaning, as determined by the programs implemented in the system (Mussio, 2003). The user identifies some subsets of pixels on the screen as functional or perceptual units, called *characteristic structures* (**cs**s). Examples of **cs**s are letters in an alphabet, symbols or icons. Users associate to each **cs** a meaning: the association of a **cs** with a meaning is called *characteristic pattern (***cp***)*. Users recognize complex **cs**s formed by more simple ones (words formed by letters, plant maps formed by icons etc.) and attribute them a meaning stemming from the meaning of the components **cs**s. The interactive system itself is interpreted as a meaningful entity, a complex **cp**.

From the machine point of view, a cs is the manifestation of a computational process that is the result of the computer interpretation a program P. (Note that words in **bold** characters denote entities perceived and interpreted by the human user, while those in courier characters denote processes and events perceived, computed and materialized by the computer). The computer interpretation of P creates and maintains active an entity, that we call *virtual entity* (ve). Actually, P is a set of programs, some of which - called I (Input) programs - acquire the input events generated by the user actions, some - called AP (APplication) program - compute the ve reactions to these events, and some - called O (output) - output the results of this computation.

The program AP (APplication program) must be defined by the ve designer, who needs to describe the ve dynamics. At each instant, the ve state is defined as a *characteristic pattern* cp=<cs, u, <intcs, matcs>>, where intcs (interpretation) is a function, mapping the current cs of the ve to the computational state u of the program AP and matcs (materialization) a function mapping u to cs.

On the human side, the user interacts with the ve in state cp, by 1- interpreting the **cs** on the screen, 2- manifesting his/her intention by an action **ac=** <operation, **cs**>, operating on the input devices of the machine such as keyboards or mice. The input program I captures the input event generated by the action **ac**, relates this event to a known characteristic structure cs, a subset of the cs on the screen, and translates it into an input to AP. AP receives these inputs and computes the response to the human activity evolving u into a new state u'. The results of this computational activity are sent to O, which materialize them as new css which modify the image maintained visible on the screen by the system. In this way the ve reaches a new state cp'.

The interaction is *adequate* if a) the **cs** recognized by the human on the screen – i.e. in the current image – matches the cs known by the system, and b) the interpretation of the human models in a plausible way the computational meaning u - i.e. the reaction of the interactive system is the one expected by the users and understandable by them.

A simple example of ve is the “floppy disk” widget to save file in the standard toolbar of MS Word™. This ve has different materializations to indicate different states of the computational process resulting from the interpretation of P: for example, once it is clicked by the user the disk shape is highlighted and the associated computational process saves in a disk file the current version of the document. Once the document is saved, the disk shape goes back to its usual materialization (not highlighted).

Virtual entities extend the concept of widgets (as the case of disk widget before) and virtual devices (Preece, 1994), being more independent from the interface style and including interface components possibly defined by users at run time. Interactive systems implemented following our approach permit new forms of tailoring, which distinguish them from traditional ones, such as Visual Basic scripted buttons in MS Word™. For example, users can add at run-time new widgets to the repertoire made available by the system. These widgets have a computational meaning defined according to the context and the task being performed. In (Costabile et al., 2002) the creation of such a ve in a medical domain is discussed: the user (a radiologist) surrounds a set of pixels tracing a closed curve defining a new **cs**, and associates an annotation to the identified area. The system recognizes the surrounded area as a cs, assigns to it a computational meaning, so defining a new ve.

An interactive system is an environment constituted by virtual entities interacting one another and with the user through the I/O devices.

The user sees the system as a whole ve, whose computational state u is materialized at each instant as an image i on the screen. The designer describes this association as a triple vs=<i, u, <int, mat>>, where i is the array of pixels constituting the current image, u is a suitable description of the current state of the process determining the reaction of the whole system to user activities, int and mat are two functions relating elements of i with components of u. This triple is called *visual sentence* (vs), and it specifies the state of the whole virtual entity (i.e. the whole system).

The designer specifies the dynamics of the system by specifying the initial visual sentence $vs_0$, the one that is instantiated when the user first accesses the system, and a set of transformation rules that specify how a vs evolves into a different one in reaction to user activities (Carrara et al., 2002b, Fogli et al., 2002).

## 6. Building SSWs in a real case

In this Section, we provide an example of applying the SSW methodology to a real case we have developed with ETA Consulting, a company producing systems for factory automation. ETA is also responsible of producing the operating software (and related user interface) for the systems that it sells.

### 6.1 Analysis

ETA Consulting has the following needs: 1) creating systems for factory automation that are usable, i.e. easy to learn and easy to use for its clients; 2) having software tools which support ETA personnel (d-experts) in the development, testing, and maintenance of such systems. As we will describe in the following, ETA personnel is composed of different categories of people with different skills, who need to perform various tasks with the software tools. In accordance with our approach, specific software environments (SSWs) must be developed for each category of users. Similarly, ETA clients need different environments specific for their tasks when operating the automation systems. The analysis we have performed with ETA d-experts and clients of ETA automation systems has lead us to foresee a SSW hierarchy structured in four levels (Figure 1):

1) *A system workshop for software engineers*. This is a basic workshop always at the top of the hierarchy since it is the one used by the team of software engineers, in which they find all tools, programming languages, etc. they need for generating the SSWs for specific applications. Using this workshop, the team defines the libraries of methods for `cs`s creation, the window system (Myers, 1995), the templates for linking `cs`s and elements of the window system, and the Interaction Visual Language, which allows also the ETA technicians (d-experts) to manage all this stuff at level 2.

2) *A system workshop for virtual entity generation*. The software engineers have created this workshop to be used together with ETA d-experts in a kind of participatory design, for generating all `ve`s necessary to the ETA d-experts to develop the systems they sell to their clients. A deep analysis of user requirements has been performed. More specifically, we have analyzed the company and the people working in the company, the kind of applications they develop, their usual clients, the notations and tools they traditionally use to develop their applications, in order to identify the interaction visual languages for this SSW. The `ve`s created in this workshop represent the tools necessary to ETA d-experts for their activities. We identified two main activities of ETA d-experts: the first one related to the software mechanical design and testing of the automation system; the second one referring to the automation system operating in the client factory (see Figure 1). Consequently, once all `ve`s are created, two child workshops are generated: the first used by ETA d-experts for creating

environments suitable for the first activity; the other for creating the applications for the clients.

3) *One or more system workshops for application workshop creation.* Given the ves made available by the system workshop at level 2, the ETA d-experts (technicians) use the workshop at this level to generate the application workshops for the other d-experts or for the end-users. They compose various prototypes of the application workshop by selecting the ves prepared at level 2. In accordance with a user-centred approach, such prototypes are evaluated together with the other d-experts and end-users in order to choose the most appropriate for them.

4) *One or more application workshops* devoted to the different professionals working at ETA, as testers (d-experts), or in the client factory, as operators of the developed application. More in detail**,** in ETA there are mechanical designers and testers, software designers and testers. Therefore, we identified for them three different application workshops: the first for mechanical testing, the second for software testing, and the last for mechanical programming of the automation systems. Besides, among ETA clients, who use the automation systems produced by ETA, we found other two kinds of users: assembly-line operators and production managers. In this case other three application workshops have been identified: one for operators and the other two for managers.

The intermediate levels in the hierarchy are developed to cope with the need to gradually adapt the systems to the complexity of the tasks (gentle slope of complexity).

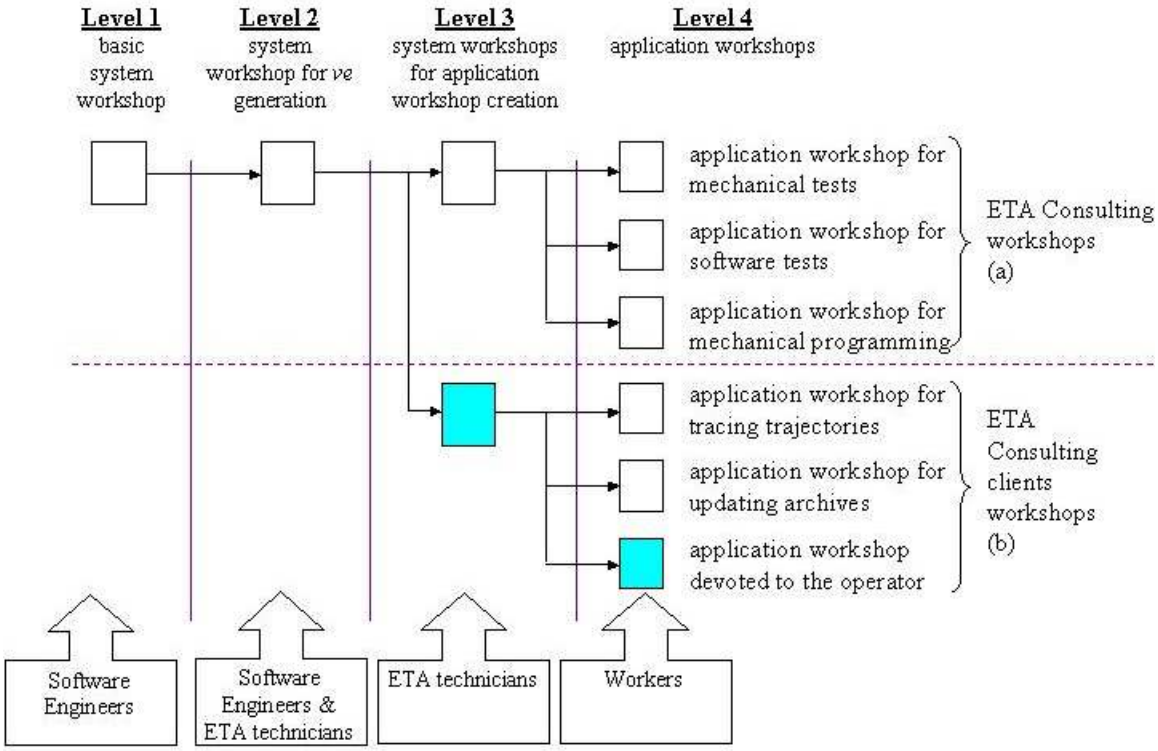



**Fig. 1**: The workshops hierarchy in the case of ETA Consulting.

### 6.2 The workshops developed for the ETA case study

At the moment, we have developed two prototype workshops for the ETA case study (Fogli et al., 2003). They are the application workshop devoted to the operator and the system workshop permitting the mechanical engineers to create the application workshop. In Figure 1, the rectangles corresponding to such workshops are highlighted.

The application workshop is devoted to the control of a pick-and-place robot. The required functionalities of this system were: different modalities of using the robot (automatic, manual, diagnostic, setting, etc.), the possibility to choose among various tools to be associated to the robot to modify its behavior and the task to perform, and finally a number of options to put annotations, to require an automatic help, to save the work, etc.

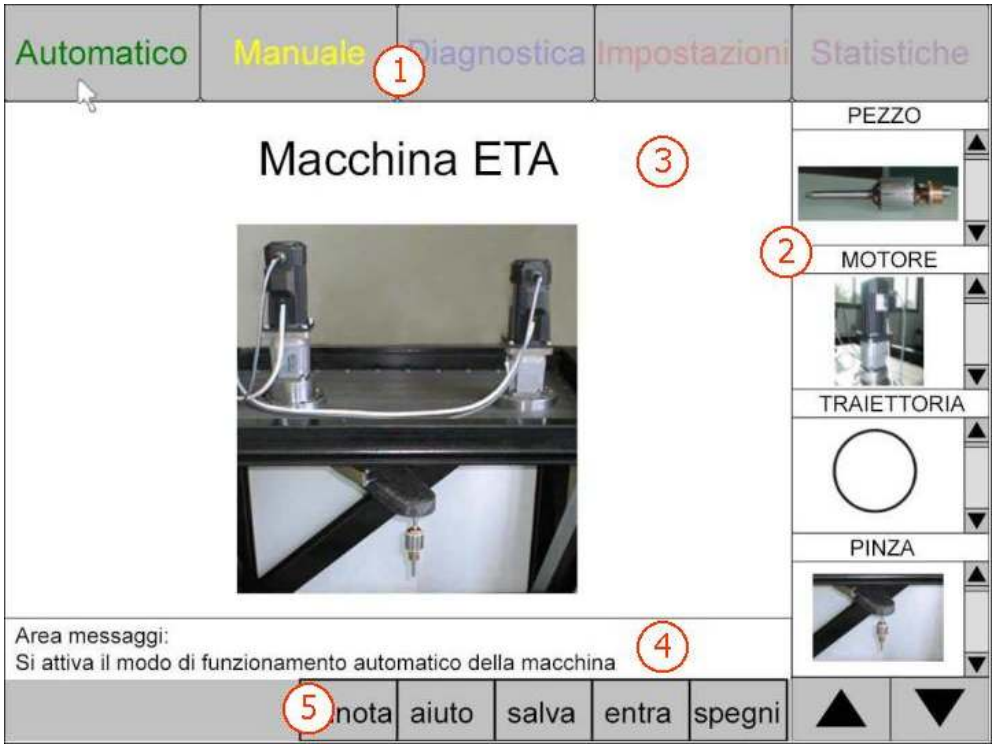



**Fig. 2**: The application workshop developed for ETA Consulting (the numbers have been added on the screenshot for the sake of explanation in the text).

A prototype of such software environment has been then developed, and its initial state is shown in Figure 2. The robot operation modality may be chosen by clicking on one of the button in the button panel indicated by the number 1 in the figure; the tools to be associated to the robot may be selected from the archives of pieces, engines, trajectories and grippers shown on the right part of the interface (2); the behavior of the machine is then shown in the working area indicated by the number 3 in the figure; finally, at the bottom, a message area (4) presents messages orienting the user during his/her interaction with the system and the button panel (5) offers the options of annotation, help, saving, logging, exit.

In the case at hand, an example of `ve` is the button “Automatico”: this `ve` has different materializations to indicate different states (characteristic patterns) of the computational process generating the `ve`: for example, once it is clicked by the user both the text and the background changes color to give a feedback to the user; moreover, the associated computational process runs the machine in the automatic modality. If the user successively clicks on another button to request a different operation modality, the colors of button “Automatico” go back to their default values (red text and gray background), the other selected button changes its colors, and the machine stops running in the automatic modality.

To create the application workshop described above, we have developed a prototypal system workshop for mechanical engineers which supports them in the creation of the application workshop through simple drag-and-drop activities. Figures 3 and 4 illustrate two different snapshots of the system workshop. The snapshots are generated during the interaction of the domain-expert with the system workshop for creating the application workshop shown in Figure 2. In Figure 3, the virtual entity “canvas” has been already selected from the menu area on the right side (1) and positioned on the working area (2) to become the background of the system that is being created. The mechanical engineer is now dragging and dropping the virtual entity “bottoniera operativa” (operative button panel). This button panel has also been selected from the menu area and can be positioned on the top of the canvas to become the area where the buttons “Automatico”, “Manuale”, etc. can be successively positioned (see Figure 2). In Figure 3, it can also be seen a message area (3), suggesting the user the operation to do in his own language, and a button panel (4) containing the tools that support the mechanical engineer in the saving or loading process, in the simulation of the system being created, in annotating the environment, and so on. Finally, a virtual entity playing the role of title (5) is also present at the top of the environment, being a cornerstone for the domain-expert.

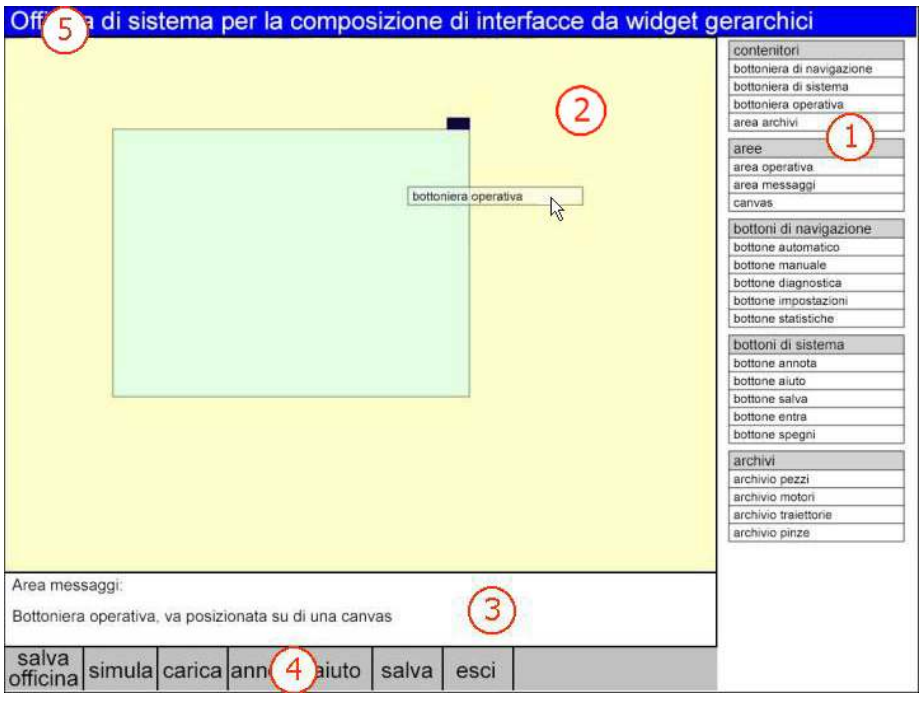



**Fig. 3**: The virtual entity “bottoniera operativa” (operative button panel) is dragged and dropped on the canvas representing the background of the workshop being created (the numbers have been added on the screenshot for the sake of explanation in the text).

Figure 4 shows the application workshop presented in Figure 2 partially composed. The d-expert is now positioning a new button on the operative button panel. Note that the virtual entity corresponding to the operative button panel is present in both snapshots shown in figures 3 and 4. However, the virtual entity is in two different states, represented by two different `cp`s: the `cp` in Figure 3 has an associated characteristic structure corresponding to the white bar with a text inside; while, in Figure 4, the characteristic structure is the gray bar over which three buttons have already been positioned.


**Fig. 4**: A part of the application workshop has been created. The button "diagnostica" (diagnostic) is being located on the operative button panel.

### 6.3 Implementation

The implementation is based on the techniques and tools made available within the W3C framework. A workshop is implemented as an IM$^2$L program. IM$^2$L (Interaction Multimodal Markup Language) is an XML-based language that provides the rules for the definition of virtual entities: its markup tags encode a description of the possible `ve`s to be used in the application at hand (Salvi, 2003). An IM$^2$L program is composed

by a set of XML-based documents and a library of javascript functions. It runs under a common web browser, enriched by the Adobe SVG Viewer plugin. SVG is the XML specification for vector graphics (W3C, 2001) and is exploited to specify the ves materialization.

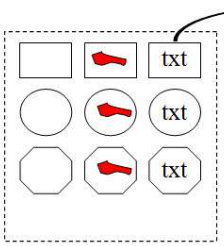
An IM$^2$L program implementing a system workshop can be steered by its users to self-transform into a new IM$^2$L program, which can results into a further system workshop or into an application workshop. This self-transformation property is used to generate a SSW hierarchy. On the whole, a SSW hierarchy is generated from the system workshop of the software engineer by an incremental process determined by the activities of the experts of the design team. More details can be found in (Fogli et al., 2003).

In order to illustrate the creation and specialization of ves necessary to the ETA environments, we describe the definition and creation of the ve "button" at the different levels of the hierarchy, by adopting the above mentioned implementation techniques.

**Level 1.** Using the SSW at level 1, the software engineer provides the IM$^2$L definition of the type "button", as shown in Figure 5. This is an XML-based description of the logical structure of a button. Most of the attributes (e.g. 'id', 'shape', 'oncl', 'onover', etc.) contain a value that will be instantiated at the next steps of the button creation (levels 2 and 3). Then, the software engineer defines a library $CS_j$ of possible button shapes as a set of SVG prototypes. A javascript function must also be defined by the software engineer to transform the IM$^2$L description of the button into an instance of the SVG prototype. Figure 6 shows an example of a library of css and the SVG prototype for a button having a rectangular shape and a textual label (the characteristic structure located at the top-right of the square). Moreover, the software engineer creates a library $U_j$ of javascript functions defining the computations to be associated to a button, including standard computations typical of a WIMP system (for example open a window when clicking on button), and application-oriented computations, i.e. related to the automation system operation in the ETA case study. Finally, the IM$^2$L definition, the SVG prototypes and the javascript functions are made available to the next level in the hierarchy (level 2).

```
<button
        template="yes"
        id="buttonIdentifier"
        position="0,0"
        dimension="0x0"
        shape="buttonShape"
        color="buttonColor"
        stroke-width="1"
        oncl="functionOnClick"
        onover="functionOnMouseOver"
        onout="functionOnMouseOut"
        <text position="0,0"
                fill="black"
                font-size="20">
                ButtonText
        </text>
        <text id="buttonIdentifierD">
                Description button
        </text>
</button>
```

**Fig. 5**: An example of IM$^2$L definition for a `ve` of type "button".




```
<g id="button"
        transform="translate(0 0)"
        flag="0"
        ondown=""
        onup=""
        onmove= ""
        onmousedown=""
        onmouseup=""
        onmousemove=""
        clone="0" drag="1" select="1">
        <rect id="" width="" height=""
                rx="0" ry="0"
                fill="rgb(0,0,0)"
        stroke="rgb(0,0,0)" stroke-
        width="1"/>
        <text transform="translate(0 0)"
        fill="rgb(0,0,0)" font-size="0"
        font-family="Arial"> </text>
        <desc id=""> </desc>
</g>
```

**Fig. 6**: The library $CS_j$ of button shapes and an example of SVG prototype for the rectangular one.

**Level 2.** At this level, the ETA d-expert associates a button shape (a characteristic structure) $cs_j$ with a computation $u_j$, by defining the pair $<int_j, mat_j>$ obtaining the characteristic pattern $cp_j=<cs_j, u_j, <int_j, mat_j>>$ that specifies the initial state of the ve "button". This association, i.e. the definition of the pair $<int_j, mat_j>$, is done by specifying the attributes in the IM$^2$L description. Some parameters are set at this level while other remain variable, to be set at the next level. As shown in Figure 7, the d-expert sets the following parameters: the button identifier, the shape of the button, the names of the computations associated with the activities performed with the mouse, and a link to a textual description of the button functionalities. All the other attributes assume default values which can be modified at level 3. Note that in the left part of Figure 7, a schematize picture of the characteristic pattern is provided, including the characteristic structure of the button "Automatico" in its initial state, the associated computations "DoAutomatico(...)", "Description(...)", "Restore(...)" (for the sake of simplicity, we do not show here the complete signatures of the functions), and the links between the characteristic structure and the computations, i.e. $int_j$ and $mat_j$.

The created characteristic patterns specifying buttons are then organized in a button library to be made available to the workshop at level 3.

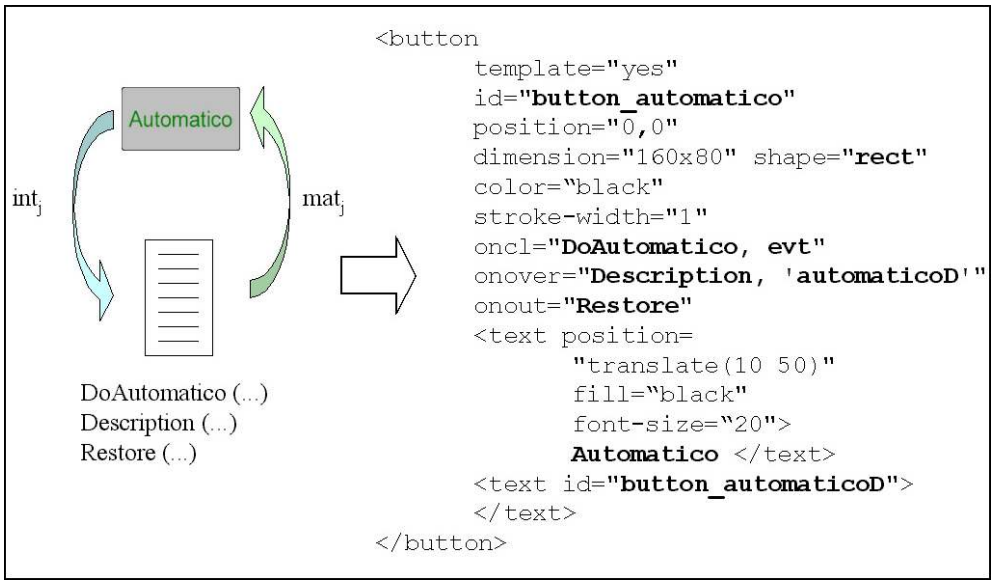



**Fig. 7**: $cp_j$ definition at level 2: values highlighted in bold are definitively assigned to the attributes.

**Level 3.** At this level, the d-expert, while composing a specific interface, instantiates the characteristic patterns made available by level 2. The interface composition is

visually performed: for example, the values of the attributes “position” and “dimension” are set automatically as a consequence of the positioning activity and the run-time adjustment of the button dimension (see also the example previously discussed with Figures 3 and 4). Other parameters, e.g. button color, can be set by the d-expert through a parameter setting facility accessible by clicking on the button. Figure 8 shows the final definition of the button: the values, which are specified at this level of the hierarchy, are in bold.

**Level 4.** At this level, the end-user uses the application workshop generated by the system workshop at level 3 to carry out his/her task. In the example case (see Figure 2), s/he may interact with the button “Automatico” to start the machine in the automatic modality. In summary, the ve button “Automatico” is incrementally defined in shape, content, and behavior throughout levels 1-3 to be used at level 4.

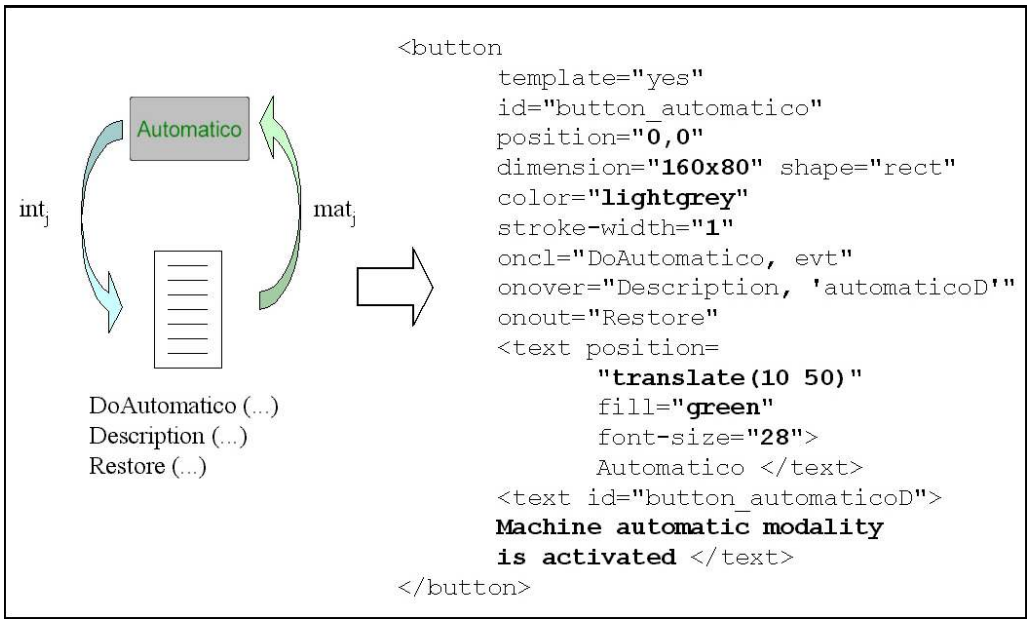



**Fig. 8**: $cp_j$ instantiation at level 3: the values in bold are definitively specified.

## 7. Conclusions

Most users require environments in which they can make some ad hoc programming activity related to their tasks and adapt the environments to their emerging new needs. Moreover, user-system interaction is currently difficult for several reasons, including the user diversity and the co-evolution of systems and users. The methodology discussed in this paper is a step toward the development of powerful and flexible environments,

with the objective of easing the way users interact with computer systems to perform their daily work. The methodology can be exploited to generate User Interface Development Environments (UIDEs) to be used by different user communities. Such environments are specialized to the community culture, background, skills and needs. It is a participatory approach, in that people belonging to a particular user community may participate to the development of the software environments devoted to the community itself. More precisely, a system workshop is a UIDE specialized to a given user community, which allows a designer to develop, generate and test SSWs devoted to other users. Whenever the developed SSW is devoted to an end-user, it is an application workshop, and it allows the user to perform an application task. If the workshop is devoted to a designer (a software engineer, an HCI expert or a domain expert), it is in turn a system workshop.

All the SSWs have the same structure. When a system workshop is used to create a new workshop, a portion of the system workshop is replicated as the kernel of the new workshop (Fogli et al., 2003).

## 8. Acknowledgements

We are grateful to Denise Salvi who developed the prototype, and to Silvano Biazzi of ETA Consulting (silvano.biazzi@cjb.it) for providing the case study.

The support of EUD-Net Thematic Network (IST-2001-37470) is acknowledged.

## 9. References

Arondi, S., Baroni, P., Fogli, D., Mussio, P. (2002). Supporting co-evolution of users and systems by the recognition of Interaction Patterns, Proc. of the International Conference on Advanced Visual Interfaces (AVI 2002), Trento, Italy, pp. 177-189.

Barnard, P., May, J., Duke, D., Duce, D. (2000). Systems, Interactions, and Macrotheory. ACM Trans. on Human-Computer Interaction, 7(2), pp. 222-262.

Boehm, B. W., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Reifer, D.J. and Steece, B. (2000). Software Cost Estimation with COCOMO II. Prentice Hall, Upper Saddle River, NJ.

Bottoni P., Costabile M.F, Mussio P. (1999). Specification and Dialog Control of Visual Interaction. ACM Transactions on Programming Languages and Systems, 21(6), 1077-1136.

Bourguin, G., Derycke, A., Tarby, J.C. (2001). Beyond the Interface: Co-evolution inside Interactive Systems. Proc. IHM-HCI 2001, Lille, France.

Carrara, P., Fogli, D., Fresta, G., Mussio, P. (2002a). Making Abstract Specifications Concrete to End-Users: the Visual Workshop Hierarchy Strategy. Proc. 2002 IEEE Symposia on Human Centric Computing (HCC'02), Arlington (VA), USA, pp. 43-45.

Carrara, P., Fogli, D., Fresta, G., Mussio, P. (2002b). Toward overcoming culture, skill and situation hurdles in human-computer interaction. Int. Journal Universal Access in the Information Society, 1(4), pp. 288-304.

Costabile, M.F., Fogli, D., Fresta, G., Mussio, P., Piccinno, A. (2002). Computer Environments for Improving End-User Accessibility. Proc. of 7th ERCIM Workshop "User Interfaces For All", Paris, France, pp. 187-198.

Costabile, M.F., Fogli, D., Letondal, C., Mussio, P., Piccinno, A. (2003). Domain-Expert Users and their Needs of Software Development. Proc. Special Session on EUD, UAHCI Conference, Crete, Greece, pp. 532-536.

Cypher, A. (1993). Watch What I Do: Programming by Demonstration, The MIT Press, Cambridge.

Dix, A., Finlay, J., Abowd, G., Beale, R. (1998). Human Computer Interaction, Prentice Hall, London.

Fogli, D., Mussio, P., Celentano, A., Pittarello, F. (2002). Toward a Model-Based Approach to the Specification of Virtual Reality Environments. Proc. IEEE International Symposium on Multimedia Software Engineering (MSE'2002), Newport Beach (CA), USA, pp. 148-155.

Fogli, D., Piccinno, A., Salvi, D. (2003). What Users See Is What Users Need, Proc. Distributed Multimedia Systems (DMS'03), Miami (FL), USA, 335-340.

ISO 5456: ISO Standard Technical Drawing Projection Methods.

Mørch, A. I., Mehandjiev, N. D. (2000). Tailoring as Collaboration: The Mediating Role of Multiple Representations and Application Units, Computer Supported Cooperative Work, 9, pp. 75-100.

Myers, B. A. (1995). User Interface Software Tools, ACM Transactions on Computer-Human Interaction, Vol. 2, No. 1, pp. 64-103.

Myers, B., Hudson, S. E., Randy, P. (2003). Past, Present, and Future of User Interface Software Tools, Human-Computer Interaction in the New Millennium, Carroll (ed.), Addison-Wesley.

Mussio, P. (2003). E-Documents as tools for the humanized management of community knowledge. Keynote Address, Proc. ISD 2003, Melbourne, AUS.

Nielsen, J. (1994). Usability Engineering, Academic Press, San Diego.

Preece, J. (1994). Human-Computer Interaction, Addison-Wesley.

Salvi, D. (2003). Progettazione di ambienti integrati per la produzione di ambienti interattivi, Laurea Thesis, Università di Brescia, Italy.

Varela, F. J. (1979). Principles of Biological Autonomy, GSR Amsterdam, North Holland, 1979.

W3C: Scalable Vector Graphics (SVG), [Online] 2001 <http: //www.w3c.org/Graphics/SVG/>.

Wulf, V. (1999). Let's see your Search-Tool! - Collaborative use of Tailored Artifacts in Groupware. Proc. of GROUP '99, Phoenix, USA, pp. 50-60.