Building Environments for End-User Development and Tailoring

Maria Francesca Costabile[1], Daniela Fogli[2], Giuseppe Fresta[3], Piero Mussio[2], 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*
{costabile, piccinno}@di.uniba.it, {fogli, mussio}@ing.unibs.it, g.fresta@cnuce.cnr.it

Abstract

Software Shaping Workshops (SSWs) described in this paper are software environments designed to support various activities of End-User Development (EUD) and tailoring. A design methodology to create easy-to-develop-and-tailor Visual Interactive Systems that are organised as SSWs is illustrated. Users of an interactive system are in many cases experts in some domain different from Computer Science, who need to perform some task with the aid of the computer system. The design methodology allows users to directly collaborate to the system design and tailoring process to face co-evolution of users and systems. The strategy feasibility is discussed, outlining its implementation through a web-based prototype.

1. Introduction

The low price of computing devices and the increasing availability of Internet connections have made a large percentage of computer users able to and interested in accessing computer-based applications. Most users have become familiar with the basic functionality and interface of computers. In addition, some results from the research in Human-Computer Interaction (HCI) and usability have started to penetrate certain product markets, thus improving the levels of usability there.

However, we believe that while some substantial progress has been made in improving the way users can access interactive software systems, some phenomena affecting the life of interactive products make difficult to develop software systems acceptable in a working environment. In HCI, it is often observed that "using the system changes the users, and as they change they will use the system in new ways" [1]. In turn, the designer must evolve the system to adapt it to its new usages; we called this phenomenon co-evolution of users and systems [2]. In [3], it is observed that these new uses of the system determine the evolution of the user culture and of her/his 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 [4], 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 (also called end-users to make explicit that they are the final users of applications developed for them) to exploit the potential benefits of task-oriented and skill-based system adaptations that only end-users 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.

Thus, 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.

The ultimate aim is empowering people 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

0-7803-8225-0/03/$17.00 © 2003 IEEE

analysis of the need of developing software that a special category of end-users, called *domain-expert users*, have; 2) a design methodology to build software environments that allow EUD activities to such users. Domain-expert users are professional people that are expert of a specific application domain and want to use computer systems for their activities, but do not have expertise of computer science or programming. The proposed design methodology is the evolution of the design strategy described in [5][6].

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 [7]. 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 disabilities. 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 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. However, there are some similarities. For instance, managing the successive versions of a piece of software is most probably a problem for software engineers as managing successive versions of documents with a word processor is a problem for end-users. Reports or letters are often written in several phases; a businessman will write successive versions of a contract that must be proofread by all parties; a home user will reuse the same letter year after year when sending his or her tax report, and just change some figures in the letter. In these cases, clever or appropriately educated users learn a simple technique aimed at helping them to manage the successive versions: assigning a number to each version. What about something of a greater complexity than the numeration of versions? One cannot expect an end-user to apply the techniques provided within the software engineering field. Software engineering methods and tools require knowledge of abstract models that end-users do not have and that require specific training. Consequently, an interesting line of research consists in identifying new sets of techniques and tools that would be the counterpart of software engineering for end-users: *software crafting*. 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 (VISs) organised as environments called Software Shaping Workshops, which will be illustrated in Section 4, is in the direction of point 3 above.

3. Emerging 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 the definition of a particular class of end-users, that we call *domain-expert users* (*d-experts* for short): they are experts in a specific domain, not necessarily experts in computer science, who use computer environments to perform their daily tasks. They have also the responsibility for induced errors and mistakes.

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 [8]. 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 [9]. 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 [4]. Tailoring can be performed only after the system has been released and therefore when it is used in the working context. In fact, the contrast often emerging between the user working activity, which is situated, collaborative and changing, and the formal theories and models that underlie and constitute the software system 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 images 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 artefacts, 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,

Table 1. Two classes of domain-experts activities, depending on whether the activity implies creating or modifying a software artefact (Class 2) or not (Class 1) [11]

class	*Activity name*	*Activity description*
Class 1	*Parameterization*	It is intended as specification of unanticipated constraints in data analysis. In this situation d-experts are required to associate specific computation parameters to specific parts of the data, or using different models of computations available in the program.
	Annotation	It is the activity by which d-experts write comments next to data and result files in order to highlight their meaning.
Class 2	*Modelling from data*	The system supporting the d-expert derives some (formal) models from observing data, e.g. a kind of regular expression is inferred from selected parts of aligned sequences [12], or patterns of interactions are derived [2].
	Programming by demonstration	D-experts show examples of property occurrences in the data and the system infers from them a (visual) function.
	Use of formula languages	This is available in spreadsheets and could be extended to other environments, such as Biok (Biology Interactive Object Kit) that is a programmable application for biologists [13].
	Indirect interaction with application objects	As opposed to direct manipulation, tools of traditional interaction styles, e.g. command languages, can be provided to support user activities.
	Incremental programming	It is close to traditional programming, but limited to changing a small part of a program, such as a method in a class. It is easier than programming from scratch.
	Extended Annotation	A new functionality is associated with the annotated data. This functionality can be defined by any technique previously described.

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 [10], 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 [10]:

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

Class 2. It includes all activities that imply some programming in any programming paradigm, thus creating or modifying a software artefact. 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.

In Table 1, we provide examples of activities of both classes from experiences of participatory design workshops in two domains, biology and earth science [11]:

4. Software Shaping Workshops

The Software Shaping Workshop (SSW) methodology we have developed to design VIS 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 to complexity. 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 [14]. 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, the methodology is aimed at generating software environments that appear to their users as workshops, providing them with the tools, organized on a bench, that are necessary to accomplish their specific activities. Users work in analogy to artisans, such us blacksmiths or joiners, i.e. users carry out their work using virtual tools that resemble their real ones. SSWs allow users to create or modify software artefacts without the burden of using a traditional programming language, but using high level visual languages tailored to users' needs. Moreover, users get the feeling of simply manipulating the objects of interest in a way similar to what they might do in the real world. Indeed, they are creating an electronic document through which they can perform some computation, without writing any textual program code.

In a SSW, users interact by using a formal version of their traditional languages and tools. In other words, the SSW approach provides each sub-community with a personalized workshop, called *application workshop*. 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 [15] and to guarantee a gentle slope to complexity.

To make clear the concepts about the SSW hierarchy, 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 VIS organized as SSW hierarchy, software engineers and d-experts have first to define the pictorial and semantic aspects of the Interaction Visual Languages (IVLs) 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 interaction it entails [16]. From this theory, we derive the formal tools to obtain the definition of IVLs.

In the PCL model, the human and the system interact by materializing and interpreting a sequence of messages at successive points in time. The human interprets the messages by applying his/her cognitive criteria, while the system applies programmed criteria. In principle, the interaction process ends when the user decides that the task has been achieved or has failed. In WIMP interaction, the messages exchanged between the user and the system are the entire images represented on the screen display, formed by texts, pictures, icons, etc. and the user can manifest his/her intention operating on the input devices of the system such as keyboards or mice. Users understand the meaning of such messages because they recognize some subsets of pixels on the screen as functional or perceptual units, called *characteristic structures* (*cs*s) [16]. The *cs* recognition results into the association of a meaning with a structure on the screen. For the system, a *cs* is a set of screen pixels to which a computational construct is associated. The designer specifies the association between a *cs* and a computational construct *u* by two functions, *intcs* (interpretation) and *matcs* (materialization), and defines a *characteristic pattern* (*cp*) as the triple $cp=<cs, u, <intcs, matcs>>$.

From the machine point of view, a characteristic structure is the manifestation of a computational process that is the result of the computer interpretation of a portion of the program specifying the interactive system. The computer interpretation creates and maintains active an entity, that we call *virtual entity* (*ve*). A simple example of *ve* is the "floppy disk" icon to save file in the iconic toolbar of MS Word™. This *ve* has different materializations to indicate different states of the computational process: 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). However, *ve*s extend the concept of widgets (as the case of disk icon before) and virtual devices [17], being more independent from the interface style and including interface components possibly defined by users at run time. The definition of virtual entities by users distinguishes our approach from traditional ones, such as Visual Basic scripted buttons in MS Word™. In [6] the creation of 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. A type is assigned to the area as a consequence of the annotation activity, and a computation is therefore associated with the *cs* by the system, so defining a new *ve*. An interactive system is thus an environment constituted by virtual entities interacting one another and with the user through the I/O devices.

A characteristic pattern specifies the state of a *ve* [18]. The user sees the system as a whole *ve*, whose computational component *u* of the state 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 whole computational process, *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.

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 to the system, and a set of transformation rules that specify how a *vs* evolves into a different one in reaction to user activities [18].

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.

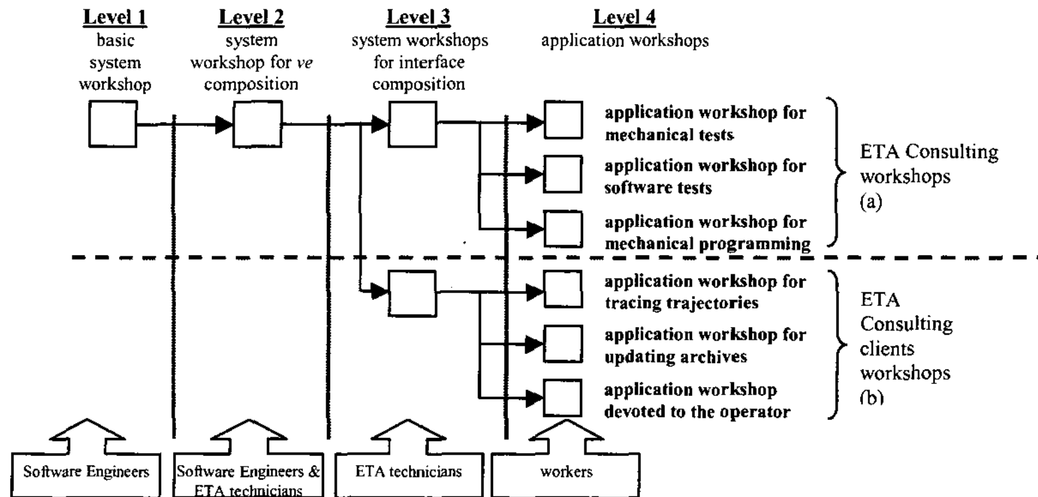

Figure 1. The hierarchy of workshops designed for the ETA Consulting case.

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 *css* creation, the window system [19], the templates for linking *css* and elements of the window system, and the IVL, which allows also the ETA technicians (d-experts) to manage all this stuff at level 2.

2) *A system workshop for virtual entity (ve) 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 *ves* 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 analysed 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 *ves* 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 *ves* 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 interface composition.* 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 interface 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

```
<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>
```

Figure 2. IM²L definition of the *ve* of type "button"

to cope with the need to gradually adapt the systems to the complexity of the tasks (gentle slope to complexity).

6.2 Implementation

The implementation is based on the techniques and tools made available within the W3C framework. The interactive environment with which the user interacts is implemented as an XML-based document and a library of javascript functions running under a common web browser, enriched by the Adobe SVG Viewer plugin. SVG is the XML specification for vector graphics [20]. The XML-based document is written adopting IM²L (Interaction Multimodal Markup Language) [21], which has been defined to specify the structure of the possible *ves* to be used in the application at hand. Lack of space prevents us from showing the user interface of each SSW. 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".

Level 1. Using the SSW at level 1, the software engineer provides the IM²L definition of the type "button", as shown in Figure 2. 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 IM²L description of the button into an instance of the SVG prototype. Figure 3 shows an example of a library of *css* and the SVG prototype for a button having a rectangular shape and a textual label. 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 IM²L definition, the SVG prototypes and the javascript functions are made available to the next level in the hierarchy (level 2).

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

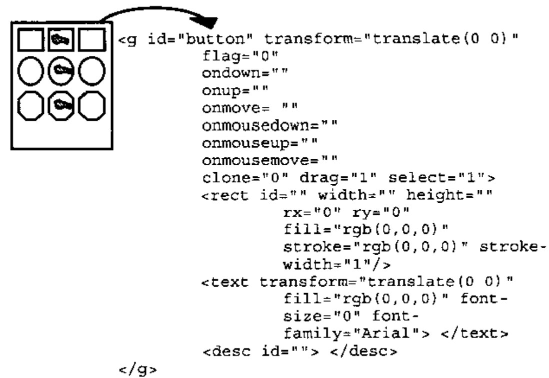

Figure 3. The library CS_j of button shapes and the SVG prototype of the rectangular shape

specifies the initial state of the *ve* "button". This association is done by specifying the attributes in the IM²L description. Some parameters are set at this level while other remain variable, to be set at the next level. As shown in Figure 4, 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. The created characteristic patterns specifying buttons are then organized in a button library to be made available to the workshop at level 3.

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. Other parameters, e.g. button colour, can be set by the d-expert through a parameter setting facility accessible by clicking on the button. Figure 5 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, s/he may interact with the button "Automatico" to start the machine in the automatic modality. In summary, the *ve* button

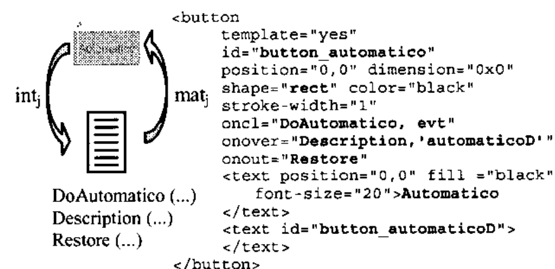

Figure 4. cp_j definition at level 2: the values in bold are definitively assigned to the attributes

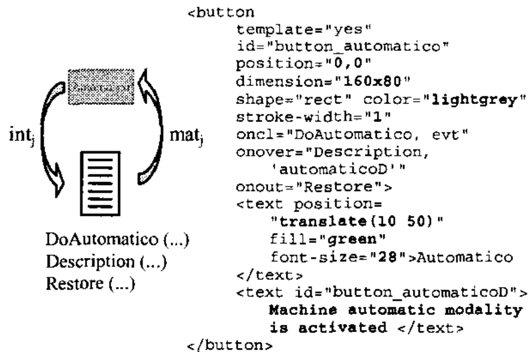

Figure 5. cp_j instantiation at level 3: attributes highlighted in bold are definitively specified

"Automatico" is incrementally defined in shape, content, and behavior throughout levels 1-3 to be used at level 4.

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.

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

[1] Nielsen, J., *Usability Engineering*, Academic Press, San Diego, 1993.

[2] Arondi, S., Baroni, P., Fogli, D., Mussio, P., "Supporting co-evolution of users and systems by the recognition of Interaction Patterns", *Proc. AVI 2002*, Trento, Italy, May 2002, ACM Press, pp. 177-189

[3] Bourguin, G., Derycke, A., Tarby, J.C., "Beyond the Interface: Co-evolution inside Interactive Systems", *Proc. IHM-HCI* 2001.

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

[5] Carrara, P., Fogli, D., Fresta, G., Mussio, P., "Making Abstract Specifications Concrete to End-Users: the Visual Workshop Hierarchy Strategy", *Proc. HCC '02*, Arlington (VA), USA, September 2002, pp. 43-45.

[6] Costabile, M.F., Fogli, D., Fresta, G., Mussio, P., Piccinno, A., "Computer Environments for Improving End-User Accessibility", *Proc. of 7th ERCIM Workshop "User Interfaces For All"*, Paris, October 23-25, 2002, pp. 187-198.

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

[8] Varela, F. J., "Principles of Biological Autonomy", GSR Amsterdam, North Holland, 1979.

[9] ISO Standard: ISO 5456 Technical Drawing Projection Methods.

[10] Wulf, V., ""Let's see your Search-Tool!" - Collaborative use of Tailored Artifacts in Groupware", *Proc. of GROUP '99*, Nov.14-17, 1999, ACM Press pp. 50-60.

[11] Costabile, M.F., Fogli, D., Letondal, C., Mussio, P., Piccinno, A., "Domain-Expert Users and their Needs of Software Development", Special Session on EUD, UAHCI Conference, Crete, June 2003, in print.

[12] Blackwell, A., "See What You Need: Toward a visual Perl for end users", *Proc. of Workshop on visual languages for end-user and domain-specific programming*, Seattle, WA, USA, September 10, 2000,

[13] Letondal, C., *Programmation et interaction*, PhD thesis, Université de Paris XI, Orsay, 2001.

[14] EUD-Net Thematic Network, http://giove.cnuce.cnr.it/eud-net.htm

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

[16] Bottoni P., Costabile M.F, Mussio P (1999) Specification and Dialog Control of Visual Interaction. *ACM TOPLAS* 21(6), 1077-1136.

[17] Preece, J., *Human-Computer Interaction*, Addison-Wesley, 1994.

[18] Fogli, D., Mussio, P., Celentano, A., Pittarello, F., "Toward a Model-Based Approach to the Specification of Virtual Reality Environments", *IEEE International Symposium on Multimedia Software Engineering (MSE'2002)*, Newport Beach (CA), USA, December 2002.

[19] Myers, B. A., "User Interface Software Tools", *ACM Transactions on Computer-Human Interaction*, Vol. 2, No. 1, March 1995, pp. 64-103.

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

[21] Salvi, D., *Progettazione di ambienti integrati per la produzione di ambienti interattivi*, Thesis, Università di Brescia, Italy, 2003.