Supporting End Users to Be Co-designers of Their Tools

Maria Francesca Costabile[1], Piero Mussio[2],
Loredana Parasiliti Provenza[2], and Antonio Piccinno[1]

[1] Dipartimento di Informatica, Università di Bari, Bari, Italy
[2] Dipartimento di Informatica e Comunicazione, Università di Milano, Milano, Italy
{costabile,piccinno}@di.uniba.it,
{mussio,parasiliti}@dico.unimi.it

Abstract. Nowadays very different people use computer systems for their daily working activities, but also for fun and entertainment or only to satisfy their information needs. Designers are doing their best to create computer systems that work as end users expect, but it must be honestly admitted that they often fail and end users have all rights to complain. In order to improve this situation and create better systems, participatory approaches have been proposed, which involve end users in the design and development process. However, this solution is not without flaws, mainly because timing and ways of users' participation are very critical. In this paper we discuss our approach to create *working systems*, which is based on a star model of the software life cycle that drives system design, development and evolution, since software design and development is seen as an evolutive process, driven by end-users activities in the real life. System development does not end with its first release; it is experimented by its end users and further evolved on the basis of their feedbacks. End users are truly engaged in the software life cycle as co-designers and experimenters of the software tools they will use in various application domains.

Keywords: Design Methodology, Star Life Cycle, Co-Evolution of Users and Systems, End-User Development.

1 Introduction

Current development of Information and Communication Technology (ICT) leads to a continuous growth of both computer systems and end-user population. Designers are doing their best to create computer systems that work as end users expect, but it must be admitted that they often fail [1]. Consequently, people are not satisfied with the system they use and companies investing in ICT are unhappy because a lot of money and resources are wasted.

In order to design successful interactive systems that meet users' expectations and improve their daily life, a designers' major issue is: "How to define an interaction language that allows end users to easily perform their activities". This language must be expressive enough to allow end users to formulate the solutions to their problems, and yet not so rich to generate user disorientation. Thus, on one side there is a

V. Pipek et al. (Eds.): IS-EUD 2009, LNCS 5435, pp. 70–85, 2009.

notation problem, while, on the other side there is a problem of system complexity. As to notation, each element of the end user-system dialog must be expressed with symbols the user can correctly interpret in her/his context and application domain. As to complexity, the language should offer to its users all and only the tools they need to perform their activities in a certain time and context.

Many questions arise. Who can define and evaluate notations understandable by users? Who can identify the set of tools required in a certain context during a certain activity? The answer is: the end users themselves. User involvement in the design team is the key point of participatory design approaches [2]. However, this solution is not without flaws, as clearly stated in [1]. Indeed, it is well known that end users are unreliable when requested to explicitly explain their needs and envision system functionalities, while they are very capable of detecting problems and difficulties when using a software system [1], [3].

We consider end users as domain experts and have worked with them in the design and development of systems in various application domains [4], [5]. Based on this experience, over the past few years we have been developing the Software Shaping Workshop (SSW) design methodology. We show in this paper that the SSW methodology is able to truly engage users at times they can provide valuable indications, as recommended in [1]. This goal is achieved by localizing the interface to user culture and allowing users to interact directly with the system under development. In this way, users are better engaged since they experiment the system in their 'sphere of work' [1]. In other words, the SSW methodology adopts a star model of the software life cycle, which anticipates the time when users test the system in their work practice [6]. This methodology acknowledges software design as an evolutive process, driven by user activities in the field. System development does not end with its first release; it is experimented by its end users and further evolved on the basis of their feedback. The system keeps evolving during time, since its use changes users' working practices, so that they require new functionalities and new tools [7].

End users are willing to be more involved in designing and developing their tools. The boom of the Web 2.0 is pushing people not only to use software, but also to create it. The Web already supports some kind of End-User Development activities, ranging from simple parameter customization to modification and assembly of components, creating simulations, games and web contents [8]. We will show in this paper how the SSW methodology supports the creation of computer systems that evolve in time and allow end users to be co-designers of their tools. In this way, it provides a contribution towards computer systems that work successfully in the real life.

The paper is organized as follows. Section 2 discusses motivations of this work. Sections 3 and 4 describe our approach to system design, development and evolution. Section 5 reports a case study that illustrates the practical application of the described concepts. Section 6 concludes the paper.

2 Background and Motivation

The diffusion of the World Wide Web as the platform for a wide variety of applications raises many expectations about the possibilities offered by web-based interactive systems. The interaction dimension creates new challenges for system specification, design and implementation. First of all, the use of an interactive system

cause the working environment and organization to evolve, and force the system to adapt to the evolved user, organization and environment (called *co-evolution* of users and systems, see [5], [9]). Moreover, current techniques for software specification and design, such as UML, are very useful for software engineers, but they are often unfamiliar to users' experience, language, and background so that they fail to provide a good communication between application designers and users. This communication gap is a reason why software systems are often poorly usable [10]. To overcome these problems, software development methodologies aiming at participatory design [2] and open-ended design [11] are invoked. However, designers must make sure that end users are engaged at opportune moments, when they can provide useful suggestions.

A further reason that makes very difficult the creation of successful systems is the diversity of end users: they have very different physical, cognitive and cultural abilities, needs, interests and activities they want to perform with computer systems. This diversity calls for general, adaptive systems [10]. The temptation is to develop very general systems, which may easily become Turing Tar Pits in which "everything is possible but nothing of interest is easy" [12]. The opposite temptation is to create specialized systems, focused on the activity of a well-specified user – or a well specified and restricted community of users linked by similar practices or similar interests – working in a restricted context. In such systems Fischer warns about the perils of this tendency: beware of the Turing Tar Pit inverse, i.e., overspecialized systems that permit only a limited number of activities, which cannot be generalized nor adapted and evolved [13]; they become a strict cage for end users by limiting their strategies for achieving their goals. Indeed, domain-specific systems support certain problem contexts but the ability to extend them is very limited; even minor incremental changes are often impossible.

The design methodology we have developed in the last few years is suitable for developing interactive systems that are not Turing Tar Pits or the inverse. The methodology stems from our experience in participatory design of several applications. However, our participatory approach is very different from the traditional one [2] that recommends to involve end users in the design team just to provide advice on their needs and expectations. They are more engaged in the overall design and development process, being not only co-designers but also experimenters of the evolving system.

Involving users in software project initiatives has been frequently indicated as a critical factor in the creation of successful software [14]. It is well acknowledged that it is good practice to involve users in designing the software applications with which they will be working. This principle of participatory design is reflected in a wide spectrum of methodologies in use today, such as agile programming.

Recent researches show that, because end users are busy with their work, they will generally not be fully engaged in analyzing and evaluating new systems [1]. They become committed only when the system impacts on their daily life, i.e. when the system is released in the field. In the design and development phases, attempts to increase user participation are helpful, but only partially effective. Our experience is in line with this view: only when a new system impacts their daily practices, end users are able to evaluate it and raise significant issues about its functionalities and usability. This does not mean that involving end users in early phases of the design process is of no value, because they certainly provide useful feedback; it suggests that

we have to revise the different stages of system development. Wagner and Piccoli recommend that post-implementation activities that try to solve the many problems raised by end users when they start working with the final system must be legitimized: they are not signs of system failure, but they are the only useful way of facing with actual users' needs and expectations [1].

One of the novelties of the SSW methodology is the proposal of modifying the traditional software life cycle by considering software design as an evolutive process, during which end users have the possibility of working in real settings with prototypes that will be evolved on the basis of the results of this work. Thus, end users are not required to *envision* since they *experience* what the end product will be and how it will impact on their work practice, being able to provide very valuable feedback.

In today information and communication society, end users are no longer passive consumers of computer tools, but they are shifting toward a more active role of information and software artifacts producers [15]. This is also highlighted by Shneiderman's claim: "the old computing was about what computers could do; the new computing is about what users can do" [16].

Our approach aims at developing software environments that support end users in performing their activities of interest, but also allow them to tailor these environments to better adapt them to their needs, and even to create or modify software artifacts. The latter are defined activities of End-User Development (EUD), to which a lot of attention is currently devoted by various researchers in Europe and all over the world [17], [18], [19], [20].

EUD implies 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. User participation in the software development process can range from providing information about requirements, use cases and tasks, as required in traditional participatory design, to creating/modifying software artefacts. 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.

To permit EUD activities, we consider a two-phase process, the first phase being designing the design environment (meta-design phase), the second one being designing the applications by using the design environment. The two phases are not clearly distinct, and are executed several times in an interleaved way, because the design environments evolve both as a consequence of the progressive insights the different stakeholders gain into the design process and as a consequence of the feedbacks provided by end users working with the system in the field. This two-phase process requires a shift in the design paradigm, which must move from user-centered and participatory design to meta-design [4], [21].

3 A Strategy for Supporting Users' Co-design

Meta-design refers to the design of environments that allow end users to be actively involved in the continuous development, use and evolution of systems. In this perspective, meta-design underlines a novel vision of system design, which is the basis of our approach and considers end users as co-designers of the tools they will

use. All stakeholders of an interactive system, including end users, are 'owners' of a part of the problem: software engineers know the technology, end users know the application domain, Human-Computer Interaction (HCI) experts know human factors, etc.; they must all contribute to system design by bringing their own expertise. Stakeholders need different software environments, specific to their culture, knowledge and abilities, through which they can contribute to shape software artifacts. They should also exchange among themselves the results of these activities, to converge toward a common design. Moreover, co-evolution of users and systems forces all stakeholders to take part in a continuous evolution of the system [6], [20]. This can be carried out, on one hand, by end users, who can perform tailoring activities to adapt the software environments they use to their evolved needs and habits. On the other hand, end users should collaborate with all the other stakeholders in the evolution of the interactive system rather than just in the original design.

Because of the diversity of end users, the challenge is to ensure the universal access and universal usability of interactive systems. The slogan "one size fits all" cannot be applied to the user interface; it is well known that people experience many difficulties when they interact with an interface presenting a huge number of functionalities, being overwhelmed with unnecessary interaction possibilities and often disoriented by them. Our approach aims at providing different communities of users with software environments that they may access and manipulate by exploiting their own system of signs (notation) [22]. We recognize, with Iverson, that a notation developed by users during years of experiences is a tool of thought [23]. However, we do not seek for a universal notation, but acknowledge that each user community has developed a notation that properly expresses the concepts and activities of that community.

The interaction language exploited in each software environment is derived from the notation used by the community the environment is devoted to. This strategy has a drawback: it makes difficult for the user to understand the improvements on the system proposed by other stakeholders. To overcome this drawback and make fruitful this clash among laguages (and cultures), the proposed approach exploits system prototypes as boundary objects, supporting the communication among the different stakeholders. Each stakeholder describes the improvement s/he wants to add to the prototype by creating an updated executable prototype and possibly annotating it. The others stakeholders receive the annotated prototype and evaluate the proposal by reading annotations and concretely experimenting the prototype in their own environment while performing their work activity, thus living concretely the experience designed by the proposer.

The *Software Shaping Workshop* (SSW) methodology we have described in [4] adopts a meta-design participatory approach that does not end with the release of the software, but continues throughout the whole software life cycle. A team of experts, including software engineers, HCI experts and domain experts, designs, implements and evolves an application throughout its life cycle. The aim of this methodology is to design interactive systems that are easily understood by their users because they "speak" users' languages. An interactive system is designed as a network of software environments, called Software Shaping Workshops (SSW or briefly workshops), each of them being either an environment through which end users perform their activities or an environment through which stakeholders participate in the design of the whole system, even at use time. An SSW is designed in analogy with an artisan or engineer

workshop, the workroom where an expert finds all and only those tools necessary to carry out her/his activities. The tools reflect the experts' needs. For example, the blacksmith's hammer is suitable for heavy work and has different features than the shoemaker's hammer, suitable for more precise work. Following the analogy, each SSW adopts a domain-oriented interaction language tailored to end-user culture, in that it is shaped and defined by evolving the traditional end-user notations and system of signs. In this sense, we refer to it as end-user language. Moreover, each SSW provides all and only those tools that are required to perform the specific activities to which the workshop is devoted. The data on which end users operate are thus represented as elements of the language. Note that using the word 'workshop' to denote the workroom we adopt the point of view of our users rather than the one of computer scientists who denotes, by this word, a brief intensive meeting.

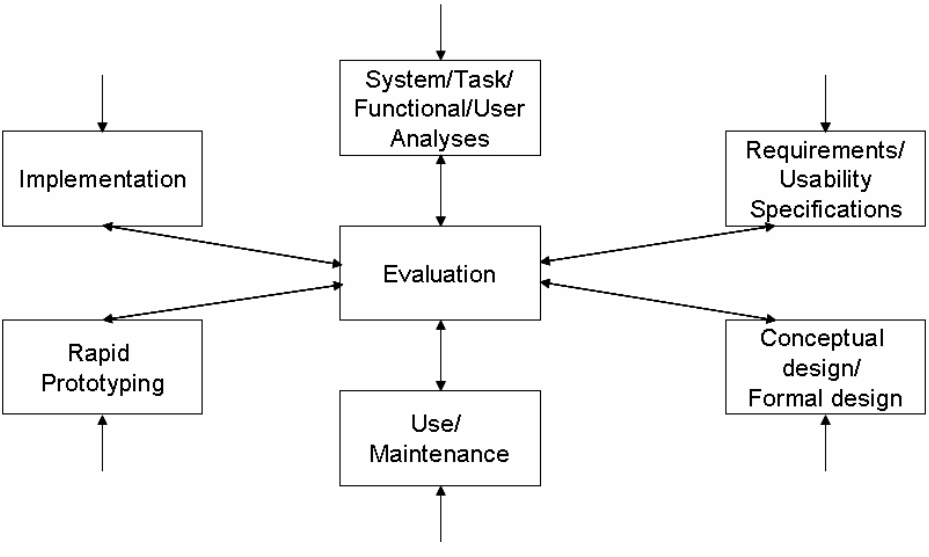

Fig. 1. The star life cycle model [1]

The SSWs are continually updated both because experience shows that the first release of a system does not generally work properly [1] and because the use of a system changes the work practice and determines user evolution (for more details on user and system co-evolution see [5]). In other words, system design and development do not end with its first release, since it evolves by following a star life cycle represented in Fig. 1 [6]. This model includes the use and maintenance activities performed during the working life of the system. The novelty of the SSW approach is that the activities in the life cycle are performed by a team including users representatives. System development can start from any point in the star (as shown by the entry arrows in the model in Fig. 1), followed by any other stage (as shown by the double arrows), always performing evaluation, which is at the center of the star. In this way, the requirements, the design and the product gradually evolve, becoming step by step well defined. The *Use/Maintenance* box refers to activities in which end

users are truly engaged; they practice in the field with the current version of the system. They can enrich the system by creating new tools and, possibly, find out new ways to use it; they also discover flaws in its use [1], [24]. All this is possible when people use the system in real life, it cannot be imagined before. New iterations of design and development are then necessary.

Results reached at each stage of the system life must be evaluated before passing to the next stage. This is why evaluation is the star center. In the SSW methodology, end users are always required to experiment the current version of the system under development: they express their observations and suggestions, resulting from such experiments. To this aim, they are allowed to annotate their own environment and to make these annotations available to the design team [25]. More details about the communications among workshops in the network, in order to evolve the system, are in [4], [5], [26]. Analogously, Software Engineering (SE) experts and HCI experts operate on prototypes and update them. The negotiation among the members of the design team is based on the use of prototypes. A modification of the system is either accepted and executed, or rejected by the team after each member has experienced it and the different findings have been discussed [25].

The SSW architecture supports the methodology: a) each stakeholder operates according to her/his mental model by using a SSW customized to her/his notations; b) prototypes (as executable specifications) and annotations, by which each stakeholder describes why and how a prototype must be updated, are exchanged among SSWs.

On the whole, the SSW methodology brings to a process of software design, development and evolution that fosters the active participation of end users, involving them when they can be more useful and productive. The process always starts with defining a prototype, which is the seed of the whole process. This prototype can be an existing system that must be improved, or a mock up that embodies the client specification, if the process starts from scratch. Each stakeholder in the design team experiments the prototype using it in her/his SSW, and finds out usability problems, or unnecessary elements, or inadequacies of the system with respect to the work organization. Each stakeholder can modify and/or annotate the prototype at hand to make explicit her/his observations. From these experiments, several proposals emerge as different improvements of the original prototype. Such different proposals are concurrently developed, subjected to a continuous experimentation and negotiation among the stakeholders, until an agreed proposal emerges. The interaction language, i.e., the set of user actions, their notation and interaction style, is progressively defined in the process, under the critical influence of the domain experts and all involved end users. As to the notation, the lexicon (textual and graphical) and the syntax are computerized versions of those used by the end users in their domain, properly enriched and formalized to be executable by a computer. The formalization process implies the careful design of the presentation elements of the user interface.

4 SSW Architecture as a Network of Customized Environments

Fig. 2 shows the SSW architecture, organized as a network of SSWs, that supports a community of end users in performing their activities as well as the design team in designing the seed of the workshops and in evolving them. The case study refers to

the development of a web application to support the activities of a consortium of small and medium-sized Italian companies operating in the confectionery field, called CIDD ("Consorzio Italiano Distribuzione Dolciaria"). More details on the case study are given in the next subsection.

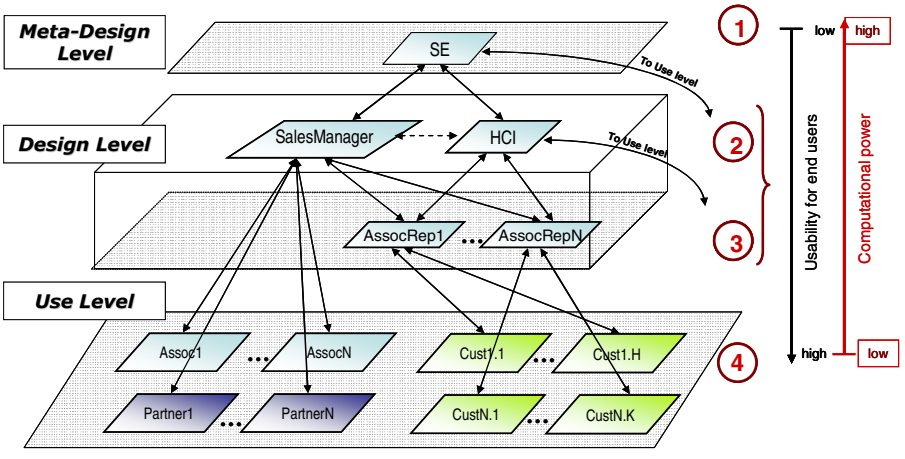

Fig. 2. The SSW network for the case study

The SSW network of an interactive system is organized in three different levels based on the different types of activities the workshops are devoted to: the *use level* includes workshops that are used by end users to perform their tasks (called application workshop); the *design level* includes workshops for designing and adapting the application workshops in accordance with the evolving knowledge and user needs (called system workshops); and the *meta-design level* includes the system workshop for software engineers, which allows them to generate and maintain all the workshops in the network.

The workshops in the architecture are of three types:

- *SE workshop* ("1" in Fig. 2): this workshop supports the software engineers in designing and evolving all the other workshops in the architecture, according to the requests of the different stakeholders. Interacting with SE workshop, software engineers perform their activities using programming languages and other development tools; they have high professional competence on software, low competence on domain activities. Even if software engineers may be considered end users when they interact with software environments, case tools, etc. created by others, in this paper end users are the domain experts for whom the system is developed and they do not have generally any expertise in computer science. The interaction languages in the SE workshop are characterized by high computational power (Turing Machine equivalent) but cannot in general be understood and managed by end users, i.e., they have low usability with respect to end users. Hence, the SE workshop would be a Turing Tar Pit for end users. In order to avoid this, more usable environments are designed for end users even if with lower computational power.

- *Design workshops* ("2" and "3" in Fig. 2): these workshops support the members of the design team other than software engineers in designing and evolving the application workshops. Such stakeholders have no (or little) competence in computer science; some of them have competence on HCI design, so that they can bring human factors in system design; other stakeholders have competence on domain activities. These team members use specialized interaction languages that have less computational power than the ones used by software engineers, in that they permit a limited set of operations. However, they are more usable for end users in that they can be correctly interpreted by end users.
- *Application workshops* ("4" in Fig. 2): these workshops are devoted to end users to perform their activities in the real world. Interacting with application workshops, end users perform their well-defined set of activities using domain-oriented languages that reflect and empower their traditional notations. Such languages permit the definition and execution of a limited set of computations, those of interest for the end-user community and are characterized by a low computational power. The characteristic structures of the alphabet elements used by an application workshop are words, icons, symbols that are significant for end users and can be correctly interpreted. Hence, the usability with respect to end users is high. Due to these languages, the application workshops are in danger of becoming the inverse of the Turing Tar Pit, where *everything is easy and very little of interest is possible*. Anyhow, this danger is avoided thanks to the support to co-evolution offered by our approach. Indeed, new functionalities can be provided to end users when they need them.

5 A Case Study

To provide a concrete example of these concepts, let us describe how the SSW methodology is being applied to the development of a web application supporting the activities of CIDD consortium. The application provides the consortium companies with several services such as price lists, discounts, order management, etc. and permits some of the consortium stakeholders to exchange information and cooperate through the Web.

After a field study, we identified the following stakeholders:

- the chairman, who is the official responsible for the consortium (e.g. he organizes and chairs meetings of associated companies, signs the balance sheet, etc.);
- the sales manager, who manages all the consortium activities;
- the consortium secretary, who works closely with the sales manager;
- the partner companies, which hold agreements with the consortium to provide goods to associated companies at special prices;
- the associated companies, which purchase products from the partner companies;
- the customers of associated companies.

A special role in the consortium is played by the sales manager, who needs to tailor the software environments to be used by the associated companies and by the partner companies, since he wants to decide about the services to provide to them. In turn, each associated company needs to define the environments to be used by their customers.

This is a typical case where various users want to co-design software environments and tools, thus the meta-design approach of the SSW methodology can be successful.

In the studies carried out for requirement analysis, four types of end users have been identified:

- *power users*: they are able to visualize, insert, modify and delete workshop contents, define access rules and even design application workshops; the role of the power user is played by the sales manager and his secretary, who works on his behalf;
- *associated companies*: their representatives can access contracts, catalogues, promotions, competitions, make orders and design/tailor the application workshops for their customers;
- *registered guests*: they are the customers of the associated companies and, through their workshops, they can visualize specific contents;
- *unregistered guests*: any user who visualizes the portal home page when browsing on the web.

The chairman is a political stakeholder, not interested in the portal use. Company customers and partners are different communities of registered guests.

A system used by different user communities is often overgeneralized for some and overspecialized for others. The SSW methodology avoids this: a system is generated as network of workshops, each one specific for tasks and needs of a user community.

The seed of the current version of the CIDD portal was a first release of the application developed as a static web site made of HTML pages that the CIDD manager had commissioned to a company. The first user of that release was the manager himself, who was not satisfied at all. The application lacked some functionalities he considered necessary and presented various usability problems; more importantly, his main complain was that he did not have the possibility of shaping the web pages for the other end users (associated companies, partner companies, etc.). In fact, one of the primary concerns of the manager is that he wants to decide the services and the functionalities of the other end users. He is a demanding user that wants to design software environments and tools used by himself and by the other users of the consortium portal.

Taking into account all manager's complains and requests, the new version of the CIDD portal was developed with the SSW methodology. By considering the different types of end users, the network architecture shown in Fig. 2 was defined. At the design level there is a workshop for HCI experts, a workshop for the sales manager and a number of workshops for representatives of associated companies ("AssocRep1", ..., "AssocRepN" in Fig. 2). The latter are used by representatives of associate companies to create and modify the application workshops devoted to their customers ("Cust1.1",..., "Cust1.H",.., "CustN.1",..., "CustN.K"). At use level there are application workshops used by associated companies ("Assoc1",..., "AssocN") and application workshop used by partner companies ("Partner1",..., "PartnerN") for their consortium activities.

Through their system workshop at the top level of the network, software engineers design and develop a first release of the system workshops for different experts (in the case study, sales manager and representatives of associated companies). By interacting with their own system workshops, such experts, who know end

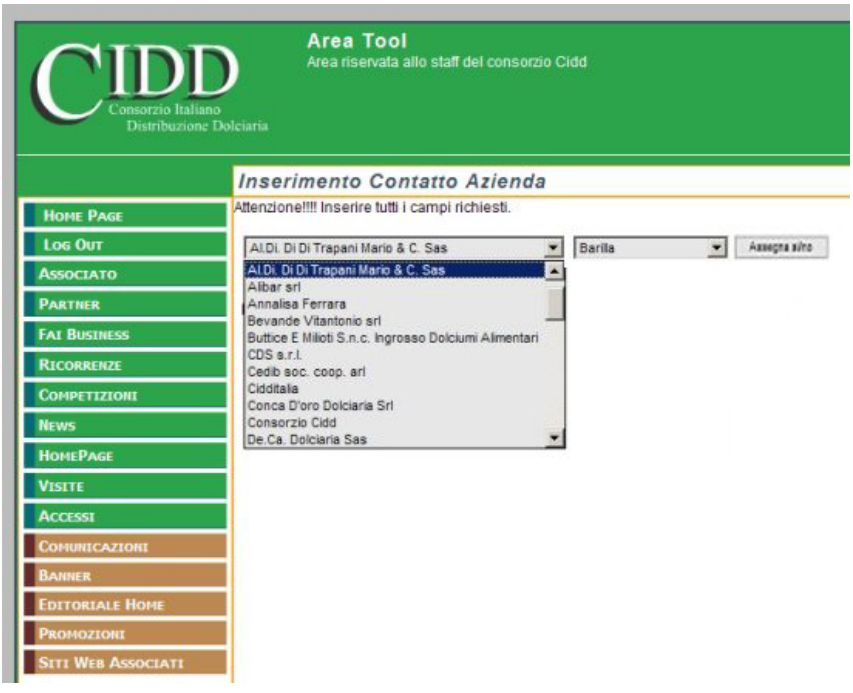

Fig. 3. The SalesManager workshop. The sales manager is generating services for associated companies.

usersworking context and habits, design and develop the application workshops tailored/specialized for different end-user communities (CIDD customers, associated companies, partners). CIDD customers, associated companies and partners use their workshops to carry out their tasks. When a user requires to perform new tasks not supported by her/his specialized workshop, s/he annotates the problems and sends it to corresponding domain expert, who evolves the workshop according to the new user requirements, by collaborating, if necessary, with HCI experts to fix usability problems. Whenever the domain expert system workshop is not so powerful to evolve another workshop, s/he asks software engineers for the missing tools by sending them an annotation. Software engineers thus evolve the workshop of the domain expert who is then able to evolve the application workshop as required.

Some examples on how end users act as co-designers are provided in the following. In the *SalesManager* workshop, the sales manager finds tools that allow him to design the application workshops for each associated company and each partner company, and the system workshops for associate company representatives ("AssocRep1",..., "AssocRepN" in Fig. 2). This workshop is shown in Fig. 3. Let us suppose that the sales manager wants to design the system workshop to be used by the representatives of an associated company, also providing it with some services. He designs this workshop by direct manipulation. Specifically, he selects the company from a drop-down list available in the central area of his workshop (the list is shown in Fig. 3); he also selects a service he wants to provide from another drop-down list

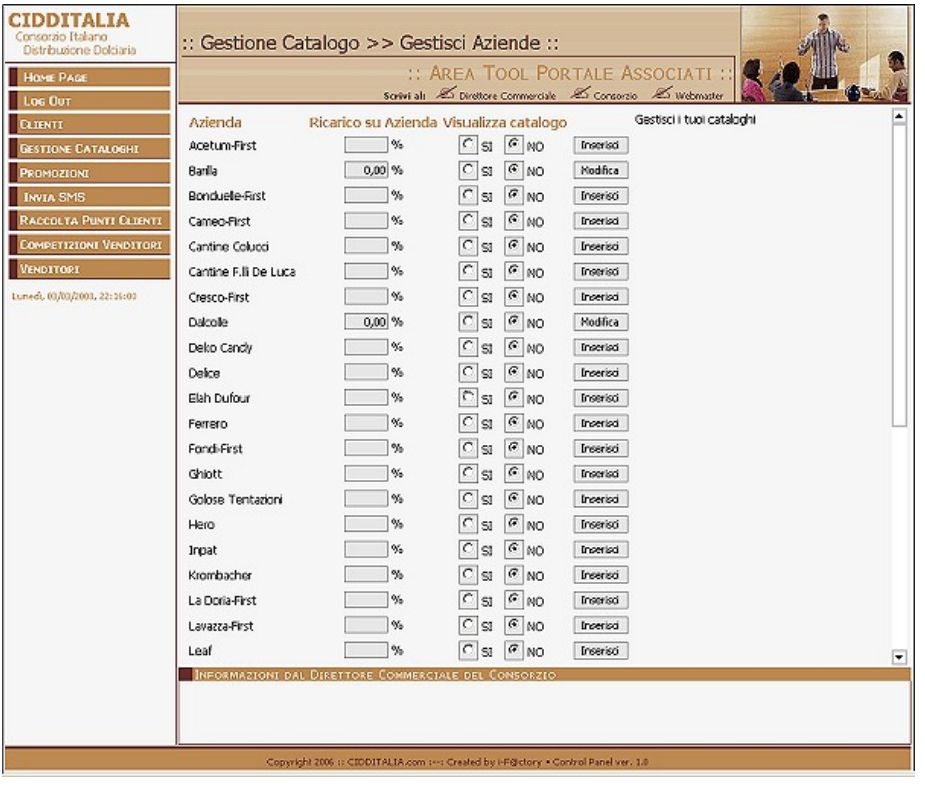

Fig. 4. A screen shot of the workshop for associated company representatives.

(available on the right of the previous one, not open in Fig. 3) and clicks on the association button (the latter on the right in Fig. 3) to actually associate the service to that company workshop. He does this for all services he wants to provide. As a result, the workshop for the representatives of the selected company is created, shown in Fig. 4. Nine services are available and they are listed in the left panel of the workshop. Fig. 4 actually shows a situation in which the user has selected a service from the left panel and the central area shows the tools available to the user for using that service. Such tools are provided through an interaction language that is suited to the culture and skills of the users, who understand the meaning of all language elements and easily work with them.

Similarly, the associated company's representatives use their workshop to design the workshops to be used by their customers (registered users). Referring again to Fig. 4, here the user defines the product catalogues for a customer, with prices and percentage of revenues, and how it can be visualized. The central area shows all companies that provide products to the customer. For each company, the user specifies in the appropriate field the percentage of revenues, and also decides whether to show prices in the catalogue or not, by clicking on a radio button. Fig. 5 shows how the catalogue is visualized in the customer workshop. Again, the user activity is specified through direct manipulation of the elements of the interaction language

Fig. 5. A screenshot of the workshop for a customer

implemented in that workshop. It is worth noting that the workshop in Fig. 4 provides its users with a communication area (the rectangular area at the bottom of Fig. 4) through which representative of associated companies can exchange messages in the network to foster the co-evolution process [5].

Through the developed system, each CCID user has available a workshop tailored to her/his needs, which allows users to interact through a domain-oriented language familiar to their culture and skills, thus avoiding the system to be a Turing Tar Pit. On the other side, users do not perceive their workshops as the inverse of Turing Tar Pits, which limit their activities, since the co-evolution process is supported throughout the software life cycle, making possible to add new functionalities, as required by end users.

We agree that the design of such complex systems requires "more knowledge than any one single person can possess, and the knowledge relevant to a problem is often distributed and controversial" [27]. The SSW methodology allows a community of stakeholders to create a system through their collaborative negotiations. This negotiation is based on the exchange of messages which are of two types: executable specifications of workshops; and annotations about these workshops. These specifications are XML-based documents [28]. A stakeholder designing or updating a workshop (the example of sales manager designing the workshop for an associated company, depicted in Fig. 3) modifies the executable specification that, when

interpreted by the browser, generates the new workshop. The user interface of this new workshop is created by:

1. the browser interpreting the document resulting from the design process;
2. the user, who can set configuration parameters (the associated company representative configures her/his workshop).

Therefore, the stakeholder designing another workshop performs a programming activity that goes beyond configuration. By configuration we intend to set parameters in order to select among functionalities available in that workshop.

In this case study, there is a variety of end users that are experts in a specific domain, but not in computer science. They need to use the web application to perform their work tasks, but they are not and do not want to become computer scientists. They are permitted to shape and modify software artefacts through interaction languages, whose elements (technical words, icon, etc.) are familiar to them. When they modify and update the CIDD application, they actually program, but they are not aware of this, also because they do not use conventional programs that would be too unfamiliar to their culture and skills. They use a language through which they compose new software artefacts by construction, similarly to children's program construction [29]. Working with these languages, CIDD users perceive that they are simply carrying out their work activities and they are highly motivated. The simplicity of the user interface is a strength of the SSW approach: "let user do simple things to generate powerful results". In other words, they are *unwitting* software developers, as analysed in [15].

6 Conclusions

This paper has discussed an approach aimed at creating interactive systems that address the needs of different communities of users, in which operations are easy to perform and many interesting activities, including end-user development activities, can be carried out. In this way, it is possible to avoid that the systems are perceived by their users as Turing Tar Pit in which "everything is possible but nothing of interest is easy". The opposite temptation is also avoided, namely the creation of overspecialized systems, in which operations are easy to perform but only specific activities, which cannot be generalized nor adapted and evolved, are possible; these systems are perceived as the inverse of Turing Tar Pits, i.e. a strict cage that limits the activities of their users.

The approach requires that an interactive system is designed as a network of software environments, called Software Shaping Workshops, through which the different stakeholders involved in system design, including end-users' representatives, are able to collaborate in the design and the evolution of the network of workshops and to carry out activities of interest in their application domain.

The SSW methodology goes beyond the traditional participatory design that has been in practice for the last two decades [2]. End users are not only involved in the design phase to provide advice on their needs and expectations, they are truly engaged in the whole process having the possibility of working in real settings with prototypes that will be evolved on the basis of their feedback. The overall software life cycle is

revised. System development does not end with its first release; it is used by people in their work practice and continuously evolved to comply with further users' needs, organization requirements and/or novel technology.

The concepts are explained through examples taken from a case study relative to the development of a web application to support the activities of a consortium of small and medium-sized Italian companies, which operate in the confectionery field.

Acknowledgments

This work was supported by the Italian MIUR and by EU and Regione Puglia under grant DIPIS. We thank the CIDD consortium and Nicola Claudio Cellamare for their collaboration in the development of the CIDD application.

References

1. Wagner, E.L., Piccoli, G.: Moving Beyond User Participation to Achieve Successful Is Design. Commun. ACM 50, 51–55 (2007)
2. Schuler, D., Namioka, A.: Participatory Design: Principles and Practices. Lawrence Erlbaum Associates, Inc., Mahwah (1993)
3. Mayhew, D.J.: The Usability Engineering Lifecycle: A Practitioner's Handbook for User Interface Design. Morgan Kaufmann Publishers Inc., San Francisco (1999)
4. Costabile, M.F., Fogli, D., Mussio, P., Piccinno, A.: Visual Interactive Systems for End-User Development: A Model-Based Design Methodology. IEEE Transactions on System Man and Cybernetics Part A-Systems and Humans 37, 1029–1046 (2007)
5. Costabile, M.F., Fogli, D., Marcante, A., Piccinno, A.: Supporting Interaction and Co-Evolution of Users and Systems. In: International Conference on Advanced Visual Interface, pp. 143–150. ACM Press, Venice (2006)
6. Bianchi, A., Bottoni, P., Mussio, P.: Issues in Design and Implementation of Multimedia Software Systems. In: IEEE International Conference on Multimedia Computing and Systems (ICMCS 1999), pp. 91–96. IEEE Computer Society, Los Alamitos (1999)
7. Nielsen, J.: Usability Engineering. Academic Press, San Diego (1993)
8. Fogli, D., Colosio, S., Sacco, M.: Managing Accessibility in Local E-Government Websites through End-User Development: A Case Study. Int. J. Universal Access in the Information Society (to appear)
9. Bourguin, G., Derycke, A., Tarby, J.C.: Beyond the Interface: Co-Evolution inside Interactive Systems - a Proposal Founded on Activity Theory. In: IHM-HCI, pp. 297–310. Springer, Heidelberg (2001)
10. Folmer, E., van Welie, M., Bosch, J.: Bridging Patterns: An Approach to Bridge Gaps between SE and HCI. Information and Software Technology 48, 69–89 (2006)
11. Hix, D., Hartson, H.R.: Developing User Interfaces: Ensuring Usability through Product & Process. John Wiley & Sons, Inc., Chichester (1993)
12. Perlis, A.J.: Special Feature: Epigrams on Programming. SIGPLAN Not. 17, 7–13 (1982)
13. Fischer, G.: Beyond Binary Choices: Understanding and Exploiting Trade-Offs to Enhance Creativity. First Monday 11 (2006)
14. Buono, P., Simeone, A.L.: An Experience About User Involvement for Successful Design. In: D'Atri, A., De Marco, M., Casalino, N. (eds.) Interdisciplinary Aspects of Information Systems Studies. Springer, Heidelberg (to appear)

15. Costabile, M.F., Mussio, P., Parasiliti Provenza, L., Piccinno, A.: End Users as Unwitting Software Developers. In: Proceedings of the 4th international workshop on End-user software engineering (WEUSE 2008), pp. 6–10. ACM, Leipzig (2008)
16. Shneiderman, B.: Leonardo's Laptop: Human Needs and the New Computing Technologies. MIT Press, Cambridge (2002)
17. Burnett, M., Cook, C., Rothermel, G.: End-User Software Engineering. Commun. ACM 47, 53–58 (2004)
18. Fischer, G., Giaccardi, E.: Meta-Design: A Framework for the Future of End User Development. In: Lieberman, H., Paternò, F., Wulf, V. (eds.) End User Development, vol. 9, pp. 427–457. Springer, Dordrecht (2006)
19. Myers, B., Hudson, S.E., Pausch, R.: Past, Present, and Future of User Interface Software Tools. ACM Trans. Comput.-Hum. Interact. 7, 3–28 (2000)
20. Sutcliffe, A., Mehandjiev, N.: Introduction. Communications of the ACM 47, 31–32 (2004)
21. Fischer, G., Giaccardi, E., Ye, Y., Sutcliffe, A., Mehandjiev, N.: Meta-Design: A Manifesto for End-User Development. Communications of the ACM 47, 33–37 (2004)
22. De Souza, C.S., Barbosa, S.D.J.: A Semiotic Framing for End-User Development. End User Development. In: Lieberman, H., Paternò, F., Wulf, V. (eds.) End User Development, vol. 9, pp. 401–426. Springer, Dordrecht (2006)
23. Iverson, K.E.: Notation as a Tool of Thought. Communications of the ACM 23, 444–465 (1980)
24. Costabile, M.F., Fogli, D., Mussio, P., Piccinno, A.: A Meta-Design Approach to End-User Development. In: IEEE Symposium on Visual Languages and Human-Centric Computing, pp. 308–310. IEEE Computer Society, Dallas (2005)
25. Fogli, D., Fresta, G., Mussio, P.: On Electronic Annotation and Its Implementation. In: Proceedings of the working conference on Advanced visual interfaces, pp. 98–102. ACM, Gallipoli (2004)
26. Carrara, P., Fogli, D., Fresta, G., Mussio, P.: Toward Overcoming Culture, Skill and Situation Hurdles in Human-Computer Interaction. Universal Access in the Information Society 1, 288–304 (2002)
27. Fischer, G.: Symmetry of Ignorance, Social Creativity, and Meta-Design. In: Proceedings of Creativity & Cognition 1999, pp. 116–123. ACM Press, New York (1999)
28. Costabile, M.F., Fogli, D., Marcante, A., Mussio, P., Piccinno, A.: A Design Methodology for Tailorable Visual Interactive Systems. In: Int. Conference on Software Engineering and Knowledge Engineering, San Francisco Bay, CA, USA, pp. 450–455 (2006)
29. Petre, M., Blackwell, A.F.: Children as Unwitting End-User Programmers. In: IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2007, pp. 239–242 (2007)