# Visual Interactive Systems for End-User Development: A Model-Based Design Methodology

Maria Francesca Costabile, *Senior Member, IEEE*, Daniela Fogli, *Member, IEEE*, Piero Mussio, and Antonio Piccinno

***Abstract*—This paper is about the development of systems whose end users are professional people working in a specific domain (e.g., medicine, geology, mechanical engineering); they are expert in that domain, but not necessarily expert in nor even conversant with computer science. In several work organizations, end users need to tailor their software systems to better adapt them to their requirements and even to create or modify software artifacts. These are end-user development activities and are the focus of this paper. A model of the interaction between users and systems, which also takes into account their reciprocal coevolution during system usage, is discussed. This model is used to define a methodology aimed at designing software environments that allow end users to become designers of their own tools. The methodology is illustrated by discussing two experimental cases.**

***Index Terms*—Design methodology, user-centered design, user interface human factors, visual languages.**

## I. Introduction

CURRENT interactive systems determine an evolution in the culture of computing and an evolution of the roles of designers, programmers, and end users in the life cycle of software products. Shneiderman [1] synthesizes this situation, claiming that "the old computing is about what computers can do, the new computing is about what people can do." To cope with this, new computing should permit humans to shape software tools to their needs, i.e., to be and act as designers in personally meaningful activities, to be and act as end users in other activities [2], or, more drastically, to "enable daily media consumer to become daily media producer" [3]. These are the problems addressed in this paper: the methodology described here is aimed at creating software systems that support end users to become designers of their tools whenever necessary for the achievement of goals chosen by themselves.

The "interaction" dimension in software systems pays much attention on the human side and creates new challenges for system specification, design, and implementation. Furthermore, it is well known that "using the system changes the users, and as they change they will use the system in new ways" [4]. These new uses of the system make the working environment and organization evolve and force the designers to adapt the system to meet the needs of the end-user organization and environment. The new interactive systems should be developed to manage this coevolution of users and systems [5].

In this perspective, we focus on the study of software environments to be used by a specific type of end users, e.g., professional people such as engineers, geologists, and medical doctors. In this paper, the word "end user" denotes experts in a specific discipline (e.g., medicine, geology), who, in general, are not expert in computer science nor are willing to be and use computer systems for their daily work activities. Our approach to system development starts from the analysis of end-user activities during their daily work and is aimed at allowing end users to exploit and evolve the system without being constrained by formalisms alien to their culture. We develop software environments that not only support end users in their specific field of activity but also allow them to tailor these environments for better adapting 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 (see [6]) and all over the world [7].

In this paper, a model of human–computer interaction (HCI) that takes into account the coevolution of users and systems during system usage is discussed. This model is used to develop a design methodology to create easy-to-develop-and-tailor visual interactive systems (VISs). Our approach stresses the importance of user diversity. End users are diverse because of their culture, education, skill, age, and training. In many domains, there are different communities of end users that need to collaborate to reach a common goal. Members of a community are, therefore, provided with an appropriate software environment that is suitable to them to manage their own view of the activity to be performed. Each environment is called a "software shaping workshop" (SSW) [8] since it is developed by exploiting the metaphor of the artisan workshop, where an artisan finds all and only the tools necessary to carry out her/his activities and properly shapes various materials (wood, iron, etc.) into usable products. By analogy, people should find in the SSWs all and only the tools necessary to shape software artifacts. Such tools must be perceived as being useful and must behave to be usable in the current situation. Overall, in our methodology, an interactive software system is developed as a network of SSWs customized to the culture and skills of

Manuscript received October 28, 2004; revised October 6, 2005 and April 13, 2006. This work was supported in part by the Italian Ministry for University and Research and by EU and Regione Puglia under grant DIPIS. This paper was recommended by Associate Editor S. Guerlain.

M. F. Costabile and A. Piccinno are with the Dipartimento di Informatica, Università di Bari, I-70125 Bari, Italy (e-mail: costabile@di.uniba.it; piccinno@di.uniba.it).

D. Fogli is with the Dipartimento di Elettronica per l'Automazione, Università di Brescia, I-25123 Brescia, Italy (e-mail: fogli@ing.unibs.it).

P. Mussio is with the Dipartimento di Informatica e Comunicazione, Università di Milano, I-20135 Milano, Italy (e-mail: mussio@dico.unimi.it).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCA.2007.904776

the stakeholders and organized in a hierarchical way. By giving insights emerging from our recent experiences, we show how the SSW methodology supports EUD.

This paper is organized as follows. Section II characterizes end users and describes the HCI model adopted for our research. Section III discusses our view on EUD. Section IV illustrates the SSW methodology. Sections V and VI present the application of this methodology to two different domains (a medical domain and a mechanical engineering domain). Section VII discusses related work, and Section VIII concludes this paper.

## II. End Users in the Interaction and Coevolution Processes

End users are persons who use computer applications as part of daily life or daily work, but are not interested in computers *per se* [9]. In this paper, the focus is on end users that are professional people working in a specific domain. Such professionals use computer environments to perform their daily work tasks and have the responsibility for possible errors and mistakes, even those that are generated by wrong or inappropriate use of computer systems. Examples are physicians [8], [10], geologists [11], technicians, clerks, analysts, and managers [12], who are increasingly required to use and even to develop software applications for their work. Having provided the definition of end user, for the sake of simplicity, in the rest of this paper, the words "end user" and "user" will be considered as synonyms.

### A. Professional People as Competent Practitioners

Before the computer age, professional people performed their tasks in *real* environments operating on *real* entities, using *real* tools, and communicating among them through *real* documents. In these working environments, professional people perform their activities as *competent practitioners*, in that "they exhibit a kind of knowing in practice, most of which is *tacit*," and they "reveal a capacity for reflection on their intuitive knowing in the midst of action and sometimes use this capacity to cope with the unique, uncertain, and conflicted situations of practice" [13].

Competent practitioners reason and communicate with each other through documents, expressed using specific notations, which represent abstract or concrete concepts, prescriptions, or results of activities. Documents expressed in the domain notations are created and interpreted using (often informally defined) alphabets and rules, which codify the explicit knowledge of the domain. They also convey what is called *implicit information*. To provide some examples, scientific communities develop "secondary notations," often never made explicit, to make their documents more readable: the use of bold characters and specific styles indicates the parts of a document—paper title, abstract, section titles—which synthesize its meaning [8], [14]. Strips of images, such as those that illustrate procedures or sequences of actions to be performed, are organized according to the reading habits of the expected reader: from left to right for western readers, from right to left for Arabic ones. Furthermore, some icons, textual words, or images may be meaningful only to domain experts: for instance, symbols representing liver cells may have a specific meaning only for hepatologists [10], whereas a radiograph may be meaningful to physicians but not to other people [8]. In other words, implicit information is significant only to people who possess the knowledge to interpret it. Most of this knowledge is not made explicit and codified; it is *tacit knowledge*, namely, it is knowledge that domain experts possess and currently use to carry out tasks and to solve problems, but that they are unable to express in verbal terms, and that they may even be unaware of. Tacit knowledge is related to the specific work domain and often to the specific situation. The use of specific notations allows domain experts to exploit their tacit knowledge; documents and messages constructed with these notations incorporate it as a part of the implicit information.

In various domains, some activities must be carried out by experts who do not constitute a uniform population but belong to different communities characterized by different cultures, goals, and tasks. More specifically, user diversity arises due to: 1) different culture, skill, specific abilities (physical and/or cognitive), and tasks to be accomplished; 2) different roles assumed by the user in performing work activities; and 3) different context of activity and geographical dispersion (of the community). For example, in the medical domain, neurologists cooperate with neuroradiologists to interpret a magnetic resonance image (MRI) and form a diagnosis; they are members of two different communities who must analyze and manage the same data set with different tools on the basis of different knowledge they possess and from different points of view. However, in this activity, as in many others, members of different communities reach a common understanding and cooperate to achieve a common purpose [15].

It is worth noting that there are domains that require years of intensive practice before practitioners achieve the most effective levels of skill [16]; end users belonging to the same community, at different levels of domain experience, express different abilities.

### B. Real and Virtual Entities

In the computer age, workplaces are augmented by the use of computer-based systems; however, professional people still perform their activities as competent practitioners. Computer systems integrate and often substitute the real tools, documents, and sometimes also the real entities on which the end users operate. The new tools, documents, and entities are *virtual* entities (ves) in that they only exist as a result of the interpretation of a program P by a computer. ves are dynamic systems that are able to capture user inputs, compute a reaction, and materialize a new state—the results of the computation—in a form perceivable by the user. For example, Fig. 1 displays a digital MRI, together with a set of widgets, which allows users to manipulate and manage it. Different from traditional MRIs, the digital MRI is an *active* document. Each pixel of the MRI can be addressed and associated with a program, a subprogram of P, which, for example, can be instantiated by the user selecting that pixel. On the other hand, the whole image disappears and is no longer accessible if P is switched off.

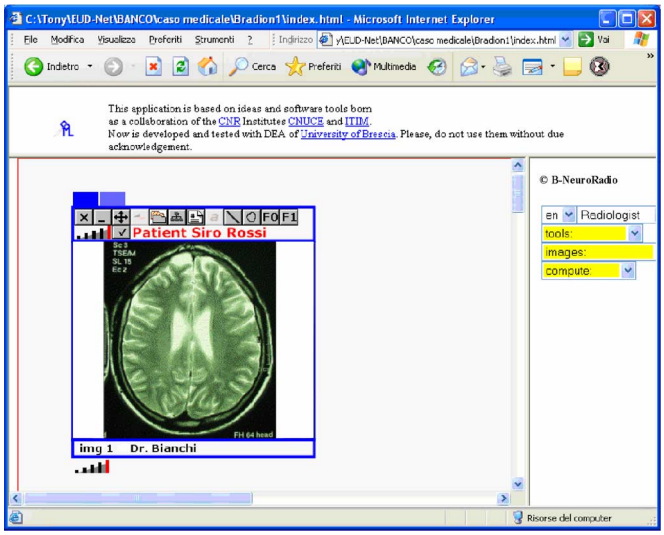



Fig. 1. Web page with the first prototype of the B-NeuroRadio software environment supporting neuroradiologists.

The appearance and the behavior of MRIs and widgets depend on P: if the parameters of P are not properly tuned, or if a different program P′ is used to interpret the data specifying the MRI and the widgets, the MRI may appear differently and/or behave in an unforeseen way. In other words, the appearance and the behavior of the digital MRI depend on the original measurement and on the program being currently used. The widgets themselves may appear and behave differently, depending on the P definition and tuning. On the whole, the window with the digital MRI, together with the set of widgets represented in Fig. 1, constitutes an example of an interactive software environment. In this case, the environment is designed and developed to support neuroradiologists in their diagnostic activities. More formally, it is a complex ve, i.e., a system of ves, in which some represent entities to be worked on, and others represent tools through which the work can be performed.

Nowadays, virtual systems of this kind are increasingly made available to end users to substitute or complement real tools and/or documents in performing end-user tasks. End users work in augmented environments operating on *real* and *virtual* entities, using *real* and *virtual* tools, and communicating among them through *real* and *virtual* documents.

### C. Modeling the Interaction Process

We model the interaction between users and computer systems as a cyclic process, in which users and systems communicate by materializing and interpreting a sequence of messages at successive points in time. These messages are subject to two interpretations: one performed by the user, depending on her/his role in the task, as well as on her/his culture, experience, and skills; and the second one is internal to the system, associating the message with a computational meaning, as determined by the programs implemented in the system. For the sake of simplicity, let us restrict the discussion to VISs based on window, icon, menu, and pointer interaction [17]; in this case, the messages that are exchanged between user and system are the whole images represented on the screen display that are formed by texts, pictures, and icons.

Users interact by operating on the input devices of the system, such as keyboards or mice. Users interpret the image that appears on the screen according to their culture, skills, and tacit knowledge. They are able to understand the meaning of the messages because they recognize some subsets of pixels on the screen as functional or perceptual units, called *characteristic structures* (**cs**s) [18], [19]. Examples of **cs**s are letters in an alphabet, symbols, or icons. Users associate each **cs** with a meaning. The association of a **cs** with a meaning is called *user characteristic pattern* (**ucp**). This association depends on the capability of the user to interpret the implicit information conveyed by the image on the screen. For example, a neuroradiologist looking at Fig. 1 may well recognize and interpret the digital MRI but, if not acquainted with Web tools, may not understand how to interact with it. On the other hand, a Web surfer may recognize several buttons, menus, and an MRI. He/she recognizes each of these **cs**s as images, describing the state of a virtual entity **ve**, with which he/she can interact. In the image, no button or menu item is selected; therefore, the surfer understands that the system is waiting for the user to decide what to do next. However, the user has to interpret the digital MRI, which is only possible for a surfer who also possesses the specific explicit and tacit professional knowledge of a neuroradiologist.

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 that stems from the meaning of the component **cs**s. The whole image on the screen is interpreted as a complex **ucp** representing the state of the VIS with which the end user is interacting.

From the computer point of view, a cs is a set of pixels generated and managed by a computational process that is the result of the computer interpretation of a program P. (Note that in the following, words in **bold** denote entities perceived and interpreted by the human user, whereas those in arial denote processes and events perceived, translated, and materialized by the computer.) The computer interpreting P creates the *virtual entity* (ve) and manages its interaction with the user (see Section II-B). A virtual entity is a virtual dynamic open system. It is *virtual* in that it exists only as a result of the interpretation of the program P by a computer, *dynamic* in that its behavior evolves in time, and *open* in that the evolution depends on its interaction with the environment. During an interaction cycle, when the user operates on some input device to manifest her/his requirement or command to the ve, the ve captures the input events generated by the user action and reacts to them by generating output events toward the user. Such output events are materialized as css on the output devices of the computer to become perceptible by the user. A cs depends on the current state u of the program P, which represents its computational meaning. In analogy with the definition of user characteristic pattern, the association of the cs with u is called the *system characteristic pattern.* A system characteristic pattern is specified as $\mathsf{cp} = \langle \mathsf{cs}, \mathsf{u}, \langle \mathsf{int}, \mathsf{mat} \rangle \rangle$, where cs is the set of pixels materialized by the program P, u is a suitable formal description of the state of P, int (interpretation) is a

function associating that cs with u, and mat (materialization) is a function associating u with cs. At each instant, the current cp describes the state of the ve in that its knowledge permits the computation of the reaction to the user input, i.e., the computation of the next state of the program P and of the new cs to be displayed, that is of the cp to be reached.

An example of a cs is  (the fifth button from the right in the toolbar above the MRI window in Fig. 1). Such a cs is the manifestation of the current state of the ve *annotation*. It is the input interface of the ve in that every user operation performed on it is captured by the program P. Moreover, it is the output interface in that every result computed by P is manifested as a new cs. This ve has different materializations to indicate different states of the computational process generating the ve: for example, once it is clicked by the user, its background changes color to give feedback to the user; moreover, the associated computational process activates an annotation window.

The whole VIS constitutes a special virtual entity. A VIS is a composed ve, characterized by the fact that in each step of the HCI process, its cs is the whole image i on the computer screen. A VIS is generated by a program P that organizes all the subprograms that bring into existence its component ves. Fig. 1 shows a whole screen image—the cs generated by a VIS at a certain step of its interaction with a neuroradiologist.

The existence of the two interpretations of a cs—one performed by the users and one performed by the computer system—explains one of the problems that arise in HCI, which is the *communication gap* between end users and designers [8]. The interpretation performed by the system reflects the designer understanding of the task at hand, implemented in the programs that control the machine. Designers develop the interactive system and primarily focus on the computational and management aspects, rather than on the solution of problems, the interaction language often being too "general" and machine oriented rather than situation and user's problem oriented. Whereas end users need to perform their tasks by reasoning in accordance to their mental models, and to express this reasoning in notations familiar to them, traditional design approaches force end users to adopt computer-oriented notations that are alien to their culture, generally not amenable to their reasoning, and often misleading for them. In this way, end users are forced to break the continuity of their reasoning to translate and express their problems and solutions into computerized language. On the whole, there is a communication gap between designers and end users, originated by their different cultural backgrounds; they adopt different approaches to abstraction since, for instance, they may have different notions about the details that can be abridged. Moreover, end users heuristically reason rather than algorithmically, using examples and analogies rather than deductive abstract tools; they document activities, prescriptions, and results through their own developed notations, articulating their activities according to their traditional tools rather than according to computerized tools. End users and designers retain distinct types of knowledge and follow different approaches and reasoning strategies to modeling, performing, and documenting the tasks to be carried out in a given application domain.

### D. Coevolution of Users and Systems

An intriguing phenomenon, often observed in HCI studies, is that "using the system changes the users, and as they change they will use the system in new ways" [4]. In turn, the designer evolves the system to adapt it to its new usages. In [20] and [21], this phenomenon is called *coevolution of users and systems*. Some seminal ideas were investigated by Mackay [22], who actually speaks about coadaptation of users and systems. Preliminary models were presented in [5] and [20].

Coevolution stems from two main sources: 1) user creativity, in which the users may devise novel ways to exploit the system to satisfy some needs not considered in the specification and design phase; and 2) user-acquired habits, in which a user may insist in following some interaction strategy to which they are (or become) accustomed (this strategy must be facilitated with respect to the initial design). An example of the first type is the integration of nonnumerical data in spreadsheets, which was included in later versions of spreadsheets, after the observation that users frequently forced the spreadsheet to manage nonnumerical data for data archiving and other tasks [4]. Other examples derive from the observation of users learning how to interact with Web documents [21].

An example of coevolution stemming from user-acquired habits is offered by the strategy for saving in a new directory a file being edited. In earlier versions of many applications (e.g., those of the MSOffice suite), after selecting the "Save as" command, the user can create a new directory, which, however, does not become the current directory. Users required a third command—open the new directory—before saving their file. In this editing situation, forcing the user to open the newly created directory is obviously inconvenient. Having recognized this contextual nuisance, more recent versions of MSOffice applications coevolved to encompass this user behavior: when a new directory is created in the "Save as" context, it automatically becomes the current one.

### E. Model of Interaction and Coevolution Processes

As a result of the considerations in Sections II-C and D, we propose an overall model that describes the two processes occurring in an augmented working environment. The first process—the interactive use of the system to perform activities in the application domain—occurs in a short time scale: every activity is the result of a sequence of interaction cycles in which the user applies her/his intuitive knowing and reflects on the obtained results, gaining new experience. The second process—the coevolution of users and systems—results in a continuous activity of revision of the system organization and tools. These two activities are cyclically repeated in a long time scale.

This novel model underlies the design methodology that supports EUD, as described in Section IV. According to this model, software engineers are required to produce interactive software systems that are able to support the two processes. In other words, such innovative systems must support end users in their work practices and also in the EUD activities that will be necessary to update the system according to coevolution.

## III. View on EUD

End users do not always perform repetitive activities; often, they are required to face unforeseen situations, in which they need to create new procedures and tools or to adapt existing procedures and tools to solve problems that cannot be foreseen in advance. End users are increasingly required to be able to produce their own software, developing software artifacts in support of organizational tasks [12]. On the whole, end users need to act as designers in some steps of their activities and as traditional users in other steps.

EUD has been defined as "the set of methods, techniques, and tools that allow users of software systems, who are acting as nonprofessional software developers, at some point to create or modify a software artifact" [6]. In EUD, tasks that are traditionally performed by software developers are transferred to end users.

To satisfy such end users' needs, an interactive system must be designed to support end users when they are acting as designers of their software tools and when they are simply using the interactive system itself. These activities last for the whole life of the interactive system because the coevolution process never ends while the interactive system is in use. As a consequence, the system life cycle starts with the design and implementation of a first version of the system, which is, thereafter, used by end users on the field. End users update the system by their EUD activities. Due to coevolution, end users also improve their knowledge, update their procedures in the real world, and modify their working organization.

To support EUD activities and still have efficient software environments, a *participatory* approach to system design is adopted here [23]. The design team includes at least software engineers, HCI experts, and representatives of end users. The latter are involved since end users are the "owners of problems" and have a domain-oriented view of the processes to be automatized. In the following, they are also called *domain-expert users* (*d-experts* for short). However, they are not expert in HCI nor in software engineering: they can only contribute to the design with their experience on the domain of activity. In turn, software engineers have the knowledge about tools and techniques for system development, and HCI experts have the knowledge on system usability and human behavior. They are necessary to the development of the system because they are the only ones who can guarantee the usability and the performance of the system. All these experts convey their experience to the design and implementation; however, none is more important than the others. All of them must recognize: 1) that each member of the team complements the ignorance of the others; 2) the need for reaching a mutual understanding; and 3) the need for peer collaboration [24].

It is worth noting that in EUD, end users play two roles: 1) as d-experts, they reason on their own working activity and design their own working environment; and 2) as competent practitioners, they perform their working activities, determining solutions to the problems in their domain and possibly adapting virtual tools to their needs. For example, a physician, as a d-expert, participates in the design of the interactive system that supports the diagnostic activity and, as a competent practitioner, uses the system to execute all the activities to reach a diagnosis in a specific case.

Once the system is in use, the design team will ideally observe end-user activities, the new usage of the system, and the new procedures induced by the evolving organization, and monitor end-user complaints and suggestions about the system in use. On the basis of these observations, the design team updates the system and sometimes also the underlying software technologies. Coevolution results, therefore, in a cyclic process, in which the usage of the system induces an evolution in the user culture and organization, which, in turn, induces an evolution of the system and of the technology. EUD must facilitate this process, allowing end users to be active partners in it. Therefore, the whole design team must remain active for the whole life of the interactive system.

The definition of EUD is thus refined as follows: EUD denotes the set of methods, techniques, and tools that allow end users to create or modify the interactive system whenever necessary and that *support the continuous coevolution of the system and its users*.

## IV. SSW Methodology

In the design methodology we propose, software environments appropriate for end users are designed in analogy with artisan workshops, i.e., environments where end users manipulate virtual entities in a way similar to artisans manufacturing their artifacts. In their workshops, traditional artisans such as blacksmiths and joiners, at each step of their activities, can extract from a repository the tools that are necessary for the current activity and set back those ones that are not useful anymore. In this way, every artisan adapts the environment to her/his needs and has available all and only the tools needed in the specific situation. By analogy, a software environment is designed as a virtual workshop, in which the end user finds a set of tools (virtual entities) whose shape, behavior, and management are familiar to her/him, being codesigned by a representative set of d-experts. Such an environment allows end users to carry out their activities and adapt their environment and tools without the burden of using a traditional programming language, but using high-level visual languages tailored to their needs. Moreover, end 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 programs, through which they later perform the necessary computations, without writing any textual program code.

Obviously, whereas traditional artisans shape real supplies, end users shape software artifacts. For this reason, we call these environments SSWs [8]. End users play two roles that must be maintained distinct, and, thus, two types of SSWs are developed. When end users perform their working activities, they use SSWs called *application workshops*, which are the results of the design activities performed by a team composed at least by software engineers, HCI experts, and d-experts. The workshops used by the design team to perform their activities are called *system workshops*. D-experts, as well as any other expert in the design team, use a system workshop customized to their culture and skills. The interactive system is, therefore,

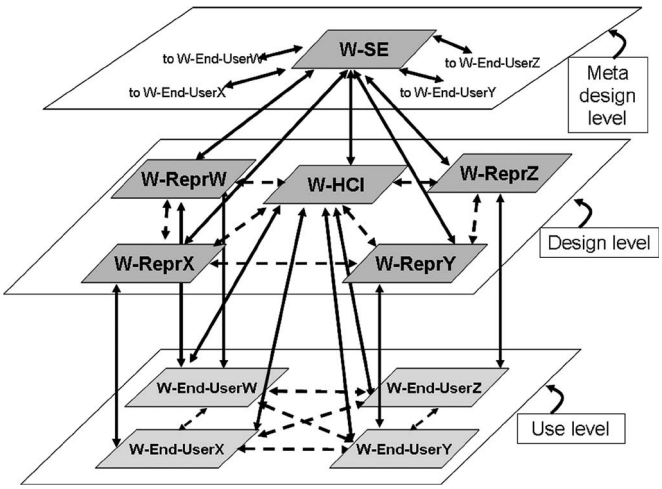



Fig. 2. General SSW network. Dashed arrows indicate communication paths, and full arrows indicate generation and evaluation paths.

designed as a hierarchical network of SSWs, each specific for a community of users.

The specification of SSWs is based on the concept of *virtual entity* introduced in Section II. More specifically, an SSW (either an application or system workshop) is a complex ve constituted by virtual entities interacting with one another and with the user through input/output devices. This ve is the result of the interpretation of the program $P_{SSW}$ defining the SSW appearance and behavior.

### A. SSW Network

The SSW network is structured so that the different stakeholders can participate in the application workshop's design, implementation, and use without being disoriented. In general, a network is organized in levels. In each level, one or more workshops can be used, which are connected by communication paths. Fig. 2 presents a generic workshop network, which include three levels.

1) Metadesign level (the top level)—software engineers use a system workshop, called W-SE, to create customized workshops to be used by other experts in the design team and to participate themselves in the design, implementation, and validation activities.
2) Design level—HCI experts and d-experts cooperate in the design, implementation, and validation activities: a design member belonging to community X participates in the design using a system workshop W-ReprX, created by the software engineers and customized to the needs, culture, and skills of community X; various experts design application workshops and also tailor their own system workshop.
3) Use level—end users of the different communities cooperate to perform a task: end users belonging to the community X participate in task achievement using the application workshop W-End-UserX customized to their needs, culture, and skills.

On the whole, both metadesign and design levels include all the system workshops that support the design team in performing the activity of participatory design. Such system workshops can be considered user interface development environments (UIDEs) [25]. The novel idea is that the UIDEs used by d-experts are very much oriented to the application domain and have specific functionalities, so that they are easy to use by d-experts.

### B. Roles in the Design Team

Software engineers are required to: 1) provide the software tools necessary to the development of the overall interactive system; and 2) participate in the design of application and system workshops. From their workshops, software engineers may reach any system and/or application workshop. In performing activities at the top level, software engineers produce the initial programs $P_{SSW}$, which generate the SSWs to be used and refined at lower levels. They participate in the design of SSWs also by modifying them to satisfy specific requests coming from lower levels. Finally, they participate in the maintenance of each SSW.

HCI experts participate in the design using their system workshop. Through this workshop, they may access all the application workshops at the bottom level to check their functionalities; they can adapt them according to the requests coming from the lower level and check their behaviors.

D-experts work at the design level by using their own system workshop to participate in the design of application workshops. Through their system workshops, they check the functionalities and behavior of application workshops and can adapt them.

The design team activity keeps going through the interactive system life cycle due to coevolution. In a first phase, called *design time*, the design team develops application workshops for the user communities addressed by the overall system. Coevolution determines the adaptation of the workshops to the new usage. Hence, in a successive phase, called *coevolution time*, the design team performs adaptation activities informed by coevolution. In these phases, HCI experts take the responsibility of the usability and accessibility aspects, and software engineers take the responsibility of the efficiency and implementation aspects. Both application and system workshops must be maintained and coevolved during the system life cycle.

### C. Communication Among SSWs

As shown by the arrows in Fig. 2, communication must be guaranteed among application and system workshops. At the use level, when working in a team, end users exchange data related to their current task to achieve a common goal. At the design level, HCI experts and d-experts exchange programs specifying workshops under construction. Whereas the d-experts bring the domain knowledge into the design, HCI experts primarily comment on the human factor aspects of the workshops that are being developed. HCI experts and d-experts also communicate with software engineers when it is necessary to forge new tools for their activities.

Communication paths exist from a lower level to the upper one. This capability is given by allowing an end user or a designer, interacting with a workshop, to annotate her/his

problems, which might depend on either lack of functionalities or poor usability, and to communicate them to all the experts reachable in the network. For example, an end user in community X, which finds some usability problems in using W-End-UserX, can annotate them and send the annotation to W-ReprX (Fig. 2). The representatives of community X, HCI experts, and software engineers can analyze this annotation, communicate among them, and agree on a possible solution to the usability problem.

More formally, software engineers at the metadesign level produce the initial programs $P_{SSW}$ and send them to the lower levels. In the current implementation, a program $P_{SSW}$ is specified as an XML-based document. As such, this specification can be annotated [26]. Later, software engineers receive from lower levels copies of programs $P_{SSW}$ annotated by the different designers and end users. They perform their activities to modify the programs $P_{SSW}$ as required and send them back.

End-user representatives, acting as designers, exchange annotated $P_{SSW}$ among themselves and with the other members of the design team for review and then send the modified programs to the use level. Since they are also in charge of SSW maintenance, they receive annotated programs $P_{SSW}$ from the use level, possibly exchanging them with HCI experts, and then send back the modified application workshops to the lower level.

Finally, end users that use a $P_{SSW}$ to perform their tasks can annotate it, or a part of it, for specific requests (to the d-experts, to the HCI experts, and/or to the software engineers) and send it to the design level.

The SSW network is modular, and the number of system and application workshops may vary according to the needs of other subcommunities of end users. The initial kernel of the network includes at least W-SE at the metadesign level, a W-Repr at the design level, and a W-End-User at the use level. End users will utilize this initial system in the workplace; later, this system will be updated according to the coevolution process.

### D. Application of the SSW Methodology

Sections V and VI will describe two cases in which the SSW methodology has been applied. The two cases refer to two contexts, which are apparently very different: the first case occurred in the medical context, in a hospital organization. The second case concerned an industrial context within a factory automation company. Although they appear distant, the two cases share methodological and structural features, which is worthwhile to highlight because they characterize the activity of end users that act as codesigners of their own systems. The two cases arisen as a result of coevolution processes in the work sites: the introduction of new technologies induced new organization of work, which, in turn, suggested the creation of new tools and procedures. In both cases, end users started developing some in-house tools and procedures. Because, usually, end users are not HCI experts or software engineers, such in-house-developed tools presented relevant usability problems. Our team of HCI experts and software engineers was then requested to revise and improve these tools. With reference to Fig. 2, our team used the same W-SE workshop to develop two different SSW networks, each one specific for an end-user community. Both networks were developed following a "star life cycle" [27], with a bottom-up participatory approach. Adopting the star life cycle means that the results of every activity are evaluated with respect to usability and feasibility before proceeding to any other activity. The approach is bottom-up in that all the prototypes were developed as evolutionary prototypes [25], which were updated on the basis of the users' requests and of the evaluation results. The approach is also participatory in that domain experts were part of the design team with a well-defined role, as described in Section IV-B. The two projects were developed in the same period; software engineers used the same tools as common root of the development, but each development was influenced by the different rhythms, styles of work, and necessities of the two organizations. However, the two processes were not independent for two reasons. The first is that the program $P_{SSW}$ has the same structure and is built from the same library of basic components, adapted to a different context. The second reason is that the findings of one project were discussed with the team of the other project. This helped in refining the methodology and, consequently, improving the tools available in the W-SE workshop.

## V. Applying the SSW Approach in a Medical Domain

To better illustrate the concepts about the SSW network, we refer here to a prototype of a VIS that is designed to support different communities of physicians, namely, neurologists and neuroradiologists, who work in different wards or different hospitals and are involved in the study of neurological diseases; they need to reach an agreed diagnosis by exchanging consultations on the cases at hand.

The improvement of the quality of the medical diagnosis is the main goal of each physician. Due to the evolution of research and technology in the medical domain, each specialist may have the aid of medical examinations of different types, i.e., laboratory examinations, X-rays, MRIs, etc. A team of physicians with different specialization should analyze the medical examinations giving their own interpretation according to their "expertise." However, the increasing number of diagnostic tools and medical specializations, as well as the increasing number of patients, does not permit the team of specialists to meet as frequently as needed to analyze the clinical cases, particularly when they work in different buildings or even in different towns or states. Information technology has the potential of overcoming this difficulty by providing software environments that allow a synchronous and/or asynchronous collaboration "at a distance." Tools for supporting the physicians to collaborate in their daily work already exist, e.g., telemedicine, videoconferencing, etc. However, physicians complain that these tools are very expensive and need large system resources, and that they are often difficult to learn and use. Some physicians more acquainted with information technologies have recently started to use cheap digital tools such as e-mail and simple image processing systems to communicate and cooperate in an asynchronous and distributed way to reach their goals. In other words, they autonomously organized their

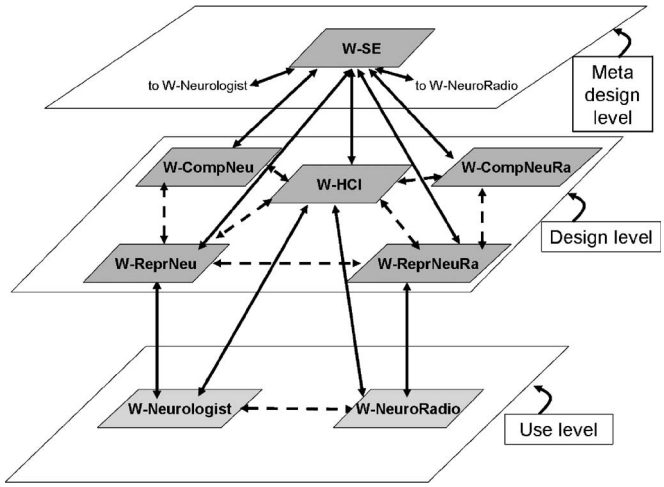



Fig. 3. Network of SSWs involved in the medical case.

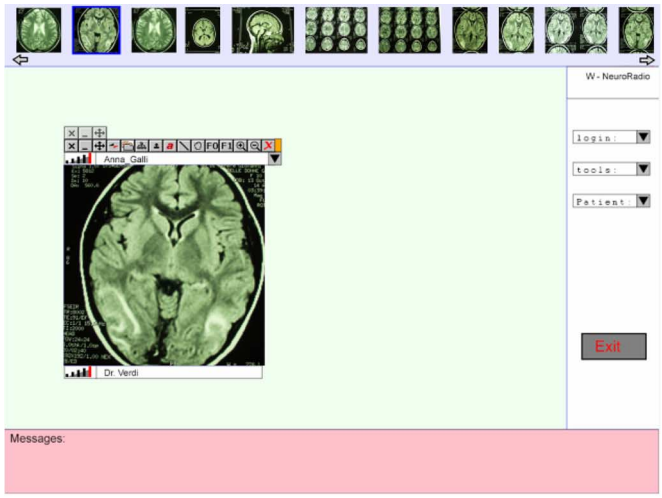
work to take advantage of the information technology, even if in a naive way. They asked us to evolve the currently used software tools to satisfy the evolved users and to fully exploit the technology for improving their work practice.

Fig. 1 shows a screenshot of the first prototype we developed and submitted to the physicians for a preliminary analysis. The prototype was used as a basis for discussing users' needs and illustrating the computer-based possibilities we were able to offer to support their work tasks. The physicians provided us with interesting feedback, and we came out together with the network of SSWs described in Section V-A.

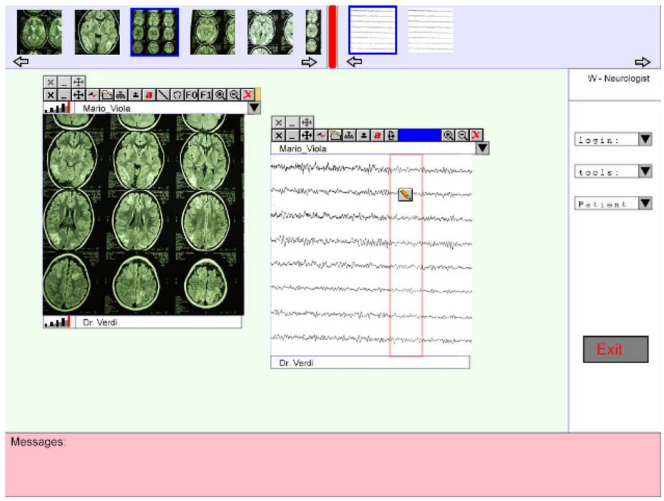
### *A. SSW Network in the Case of Neurologists and Neuroradiologists*

The analysis of the work organization in this domain led us to design the SSW network presented in Fig. 3. At the use level, neurologists and neuroradiologists use application workshops, i.e., W-Neurologist and W-NeuroRadio, respectively, which are tailored to their culture, skills, and needs, in an asynchronous and distributed way. Since neurologists and neuroradiologists possess the knowledge about the language and notation of the specific community they belong to, at the design level, their representatives (d-experts) are provided with two system workshops (W-ReprNeu and W-ReprNeuRa) to generate and maintain the application workshops, and with two system workshops (W-CompNeu and W-CompNeuRa) to generate and maintain components to be used in the other system workshops for their design activities. A fifth system workshop, i.e., W-HCI, is used by HCI experts to check the human factor aspects of the generated application workshops. HCI experts and d-experts collaborate with software engineers (who use the system workshop W-SE at the metadesign level) in a participatory design process, bringing their own knowledge and expertise to create and evolve the two application workshops used by the two communities of end users (neurologists and neuroradiologists).

The workshops in the network support the collaboration of users at each level of the hierarchy. In particular, at the use level, W-Neurologist and W-NeuroRadio help the different specialists to achieve a diagnosis for a particular case. This




(a)




(b)

Fig. 4. Two application workshops. (a) Application workshop for neuroradiologists. (b) Application workshop for neurologists.

collaboration is obtained by performing EUD activities at the use level [8] and exploiting the tools for electronic annotation described in [26]. These workshops allow the specialists to cooperate in virtual asynchronous meetings. The specialists may use their own application workshops to perform their daily work tasks: for example, a specialist may analyze the available electroencephalogram (EEG) or MRI, perform annotations and/or computations on them, select parts of them, and define diagnoses and/or consultation requests. Each workshop is equipped with a certain set of tools that are necessary to carry out the work tasks and are customized to the specific user community. For example, both application workshops devoted to neurologists and neuroradiologists have an overview area on the top of the screen, which may be used to browse MRIs or EEG portions. The overview area is the electronic counterpart of the diaphanoscope, which is used by the physicians in a real meeting to analyze MRIs. Since neuroradiologists are only interested in MRIs, in their application workshop, they find only the MRI overview area [see Fig. 4(a)] together with tools to process MRIs and to formulate diagnoses. On the other hand,


Fig. 5. Neuroradiologist is accessing the request of consultation received from the neurologist. The text of the annotation is magnified here for easy reading.


Fig. 6. Neuroradiologist has identified a suspect area in the MRI and has surrounded it. From the "tools" menu, he is now selecting a tool to automatically compute some values relative to the hyperdensity of the suspect area.

neurologists primarily study a great number of EEGs; however, in some cases, they analyze MRIs. Thus, in their application workshop, there are two overview areas, which are resizable [see Fig. 4(b)], so that the neurologists can reduce (or even completely close) the area containing the MRIs to expand the EEG overview area according to their needs.

The two application workshops have similar layouts and interaction possibilities: they present the overview area at the top, a work area in the center, a menu area at the right side, and a message area at the bottom to provide the user with some explanations when necessary. In particular, from the menu area, users can choose the login type (e.g., senior or practicing physician), the available tools (e.g., a tool for sending a request to a colleague, a tool for image processing, etc.), and the patient to study.

The SSW network is modular, and it is possible, in the future, to extend it by creating other SSWs for other stakeholders, e.g., other physicians or clerks and managers dealing with management and billing systems.

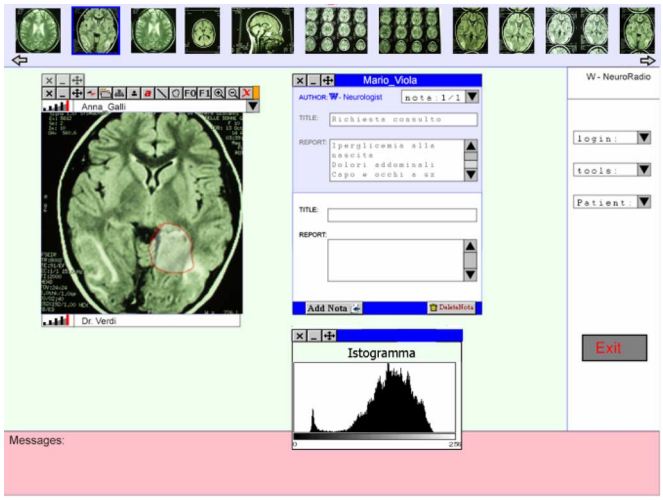



Fig. 7. By clicking on the suspect area, the neuroradiologist has obtained a histogram of the hyperdensity.

### *B. EUD at the Use Level*

Let us discuss some examples of EUD at the use level. The application workshop prototypes in Fig. 4 improve our first prototype (Fig. 1). The improvement derives from the observations collected during a field study. We observed that physicians, when examining the images under study during their meetings, point to structures of specific interest and make comments on them. Other users, in other domains, have a similar behavior. We then created suitable annotation tools that permit associating new widgets with particular structures identified in the images [8], [11]. This is an EUD activity. To see how this works in W-Neurologist and W-NeuroRadio, let us consider, for example, that a neurologist has requested a consultation to the neuroradiologist. He did this by annotating a specific MRI with information that is useful to the neuroradiologist and sending him a request for consultation. Fig. 5 shows W-NeuroRadio once the neuroradiologist is examining the MRI annotated by the neurologist. The annotation is shown in the window to the right of the MRI. The part of the annotation window that reports indications about the patient symptoms, provided by the neurologist, is magnified in Fig. 5 for better reading.

The neuroradiologist studies the MRI and identifies a suspect area, surrounding it by using the tool for curve tracing (the tool is activated through one of the icons on top of the MRI). Then, he accesses the tools for image processing provided in the "tools" menu of W-NeuroRadio (Fig. 6). The physician selects "iperdensità" ("hyperdensity") to compute a histogram of the hyperdensity values and other parameters relative to the suspect area. This new information is automatically associated with the suspect area. A new widget is then created whose characteristic structure is the set of pixels of the suspect area; the user can click on it whenever s/he wants to display the associated information. Fig. 7 shows W-NeuroRadio once the user has selected the new widget and displayed the histogram to study the hyperdensity of the suspect area.


Fig. 8. W-Neurologist visualizes the hyperdensity basic parameters associated with the circled area in the MRI.

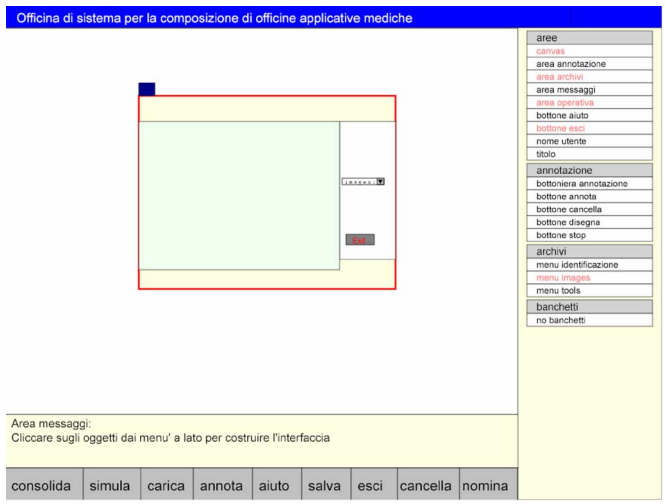



Fig. 9. System workshop W-ReprNeuRa during the interaction with a d-expert to create the application workshop devoted to neuroradiologists.

The neuroradiologist replies to the neurologist's consultation request by providing his opinion that he writes in the bottom area of the annotation window (the one that is empty in Fig. 7). The whole text becomes a new annotation associated with the suspect area in the MRI, and it is saved in the repository shared by W-NeuroRadio and W-Neurologist.

The neurologist, using W-Neurologist and accessing the same MRI, finds the updated annotation with the answer of the neuroradiologist and a new widget on the MRI, namely, the circled area to which the data evaluated by the neuroradiologist are associated. This widget, being accessed through W-Neurologist, behaves according to the neurologist culture: when the neurologist selects it, the workshop reacts by presenting the associated data in a representation understandable by the neurologist. In the example of Fig. 8, basic parameters associated with the hyperdensity of the circled area (the suspect area) are shown instead of the histogram shown in W-NeuroRadio. This is because the histogram would be difficult to interpret by the neurologist.

The creation of a new widget through annotation is a tailoring activity performed by end users.

### C. EUD at the Design Level

Each workshop is the result of a participatory and cooperative design performed by a team including d-experts (e.g., senior neurologists or neuroradiologists), HCI experts, and software engineers. To this end, representatives of end users, i.e., d-experts, interact with a system workshop to create the application workshops by direct manipulation of virtual entities prepared by software engineers and HCI experts. As we said in Section IV-A, system workshops used by d-experts are a type of UIDE customized to d-experts and to their context and tasks to improve their ease of use.

Fig. 9 shows a screenshot of the system workshop W-ReprNeuRa: the d-expert has partially composed the application workshop shown in Fig. 4(a) devoted to neuroradiologists by selecting, from the repositories on the right side of the screen, the canvas on which the application workshop is composed, the overview area, the work area, the archive area, the Exit button, and the "Patient" menu. The d-expert is creating a program by visually interacting with the system workshop and is not writing any textual code.

The tools available to the different specialists are also customized to their needs. For example, in their daily work, neurologists and neuroradiologists study parts of EEGs and/or MRIs in depth. As shown in Fig. 4(a), the neuroradiologist has selected one part of the MRI from the MRI overview area, which has been automatically loaded in a specific window, called a *workbench*, to be studied and manipulated by the user. Similarly, in Fig. 4(b), the neurologist has selected one portion of an EEG from the EEG overview area and one of the MRIs from the MRI overview area. Each image has been loaded into a different workbench, and the neurologist can study and manipulate them separately. In both cases, the workbenches are customized to the image to be processed and to the needs of the physician: they are equipped with a toolbar hosting the tools to be used by the physician to study the image and prepare requests of consultation to be sent to other specialists. According to the principles of the SSW methodology, these tools resemble the real tools the physicians use in their work practice.

Two further system workshops are used by representatives of end users to prepare customized workbenches. Fig. 10 shows the system workshop called W-CompNeuRa used by a senior neuroradiologist for creating the workbench to be used by neuroradiologists to study MRIs. With the aim of not disorienting the user, this system workshop has the same layout organization and interaction possibilities of the system workshop to be used to create whole application workshops: it presents a title on the top, a work area in the center, a set of repositories on the right, a message area, and a button panel on the bottom. The user can select objects from the repositories and drag and drop them in the work area to create, in this case, a workbench. After a workbench is saved, it appears as one of the benches available in the corresponding repository in the system workshop

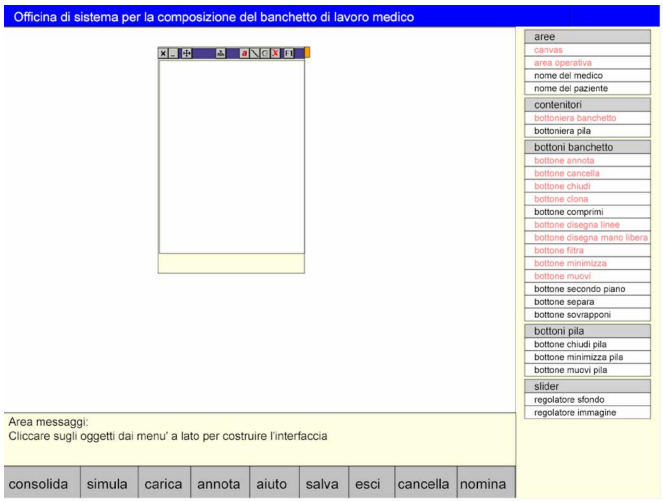



Fig. 10. System workshop W-CompNeuRa during the interaction with a senior neuroradiologist to create the workbench for the neuroradiologist to study MRIs.

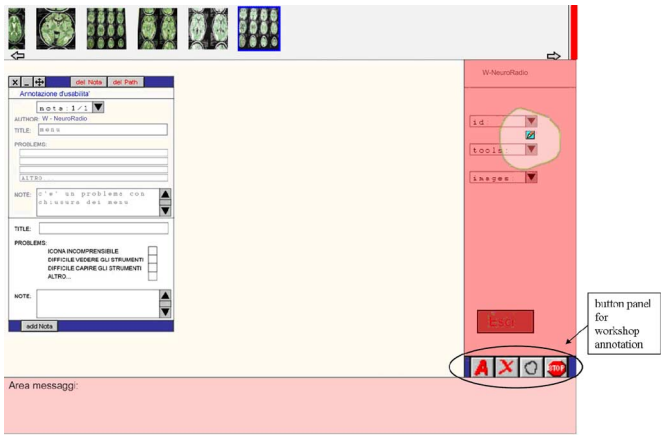



Fig. 11. Neuroradiologist is annotating the application workshop W-NeuroRadio.

W-ReprNeuRa presented in Fig. 9. It may, thus, be selected to be added to the application workshop being created.

### *D. Communications in the Case of Neurologists and Neuroradiologists*

With reference to the SSW network illustrated in Fig. 3, communications occur within the same level and between different levels of the network. At the design level:

1) Senior physicians exchange with HCI experts the application workshops they are developing through W-ReprNeu or W-ReprNeuRa to evaluate and improve their usability; in this case, programs $P_{SSW}$ are exchanged.
2) Senior physicians exchange with HCI experts components of SSWs (e.g., subprograms that, when interpreted, generate virtual entities of type "workbench") they are developing through W-CompNeu or W-CompNeuRa to evaluate and improve their usability.
3) Senior neurologists interacting with W-ReprNeu to create W-Neurologist can access the tailored components developed using W-CompNeu; also, in this case, programs $P_{SSW}$ are exchanged.
4) Senior neuroradiologists interacting with W-ReprNeuRa to create W-NeuroRadio can access the tailored components developed using W-CompNeuRa; therefore, also in this case, programs are exchanged.
5) Senior neuroradiologists interacting with W-ReprNeuRa exchange programs and data with senior neurologists interacting with W-ReprNeu to decide on the types of tools for consultation and data representation they would like to place at neurologists' and neuroradiologists' disposal in W-Neurologist and W-NeuroRadio.

At the use level, neurologists and neuroradiologists exchange data (textual or visual annotations, images) with the aim of achieving a common diagnosis.

Communications between different levels may occur from top levels to bottom levels and vice versa. In the first case, programs specifying whole workshops are exchanged. For example, d-experts create and send application workshops to end users. In the latter case, annotations about usability problems or new user needs are sent to the higher levels: for example, at the design level, representatives of end users can send to software engineers requests for developing new tools to be used in their workshops; whereas at the use level, end users can communicate to HCI experts or to d-experts their problems in understanding the meaning of some data representations or how to use some tools. We describe this second case in more detail. Users interacting with an application workshop can choose the "annotation mode" from the "tools" menu and, thus, add textual and graphical annotations about usability problems affecting the software environment with which they are working. Fig. 11 shows a screenshot taken during the annotation of W-NeuroRadio on behalf of the neuroradiologist.

In the annotation mode, the user cannot perform her/his task (in this case, analyzing and annotating MRIs) but can interact with the workshop only to annotate it. To this end, a button panel, appearing only in annotation mode (see Fig. 11), provides tools that permit inserting textual annotations, deleting annotations, designing closed curves over critical portions of the workshop, and stopping the annotation mode.

By clicking on a portion of the screen, the user can highlight a component of interest: as can be seen in Fig. 11, a colored layer has appeared over the "menu area." Then, by selecting the "tracing closed curves" button, the user has highlighted an area over the first two menus. Then, he has annotated in the annotation window the problem related with these menus.

## VI. Applying the SSW Approach in a Mechanical Engineering Domain

We describe here a further example of an SSW network, with reference to a prototype we have developed with a factory automation company. The company is responsible for producing the operating software (and related user interfaces) for the systems it sells to client companies. In the rest of this section, by "company," we refer to the factory automation company and by "client companies," to its clients. The company has the following needs: 1) to create systems for factory automation

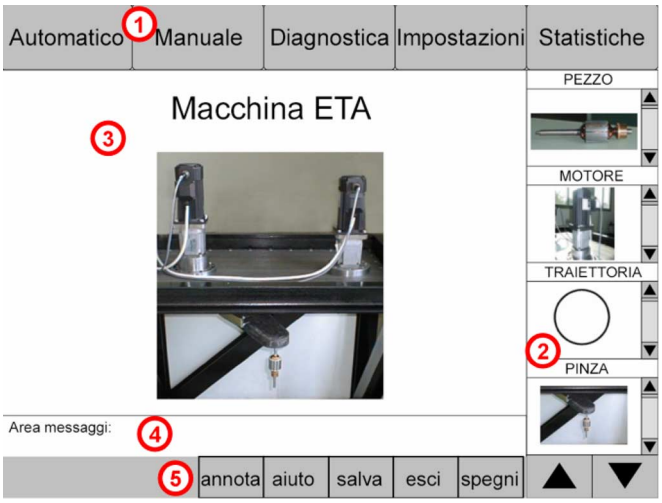



Fig. 12. Application workshop developed for a factory automation company (the numbers have been added on the screenshot for the sake of explanation in the text).

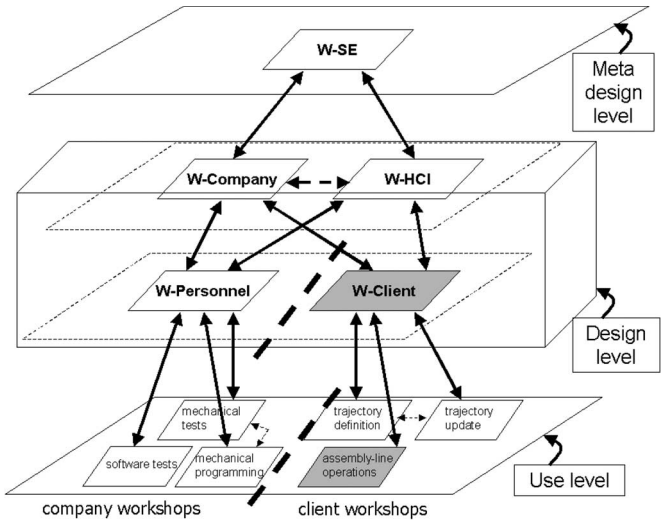



Fig. 13. SSWs in the case of factory automation.

that are customized and usable for their client companies, i.e., easy to learn, use, and adapt for experts in factory automation (but not in computer science); and 2) to create software tools that support its personnel in the development, testing, and maintenance of such systems. The company personnel are organized into different categories of end users with different responsibilities and skills, who need to perform various tasks with the software tools. According to the SSW methodology, specific software environments must be developed for each community of end users. Similarly, the client companies need different environments specific to their own organization and for their tasks when operating the automation systems.

The analysis we performed with company personnel and client companies led us to design a VIS that can be shaped to the specific client company and structured in various environments, each for a specific community of users. As an example, Fig. 12 shows the prototype developed for the assembly line operator, which is devoted to the control of a pick-and-place robot. The required functionalities of this environment are different modalities of using the robot (automatic, manual, diagnostic, setting, etc.); the possibility to choose among various tools to be associated with the robot to modify its behavior and the task to perform; and, finally, a number of options to create annotations, get online help, save the work, etc. The robot operation modality is chosen by clicking on one of the buttons in the button panel indicated by the number (1) in Fig. 12; the tools to be associated with 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 work area (3); at the bottom, a message area presents messages orienting the user during her/his interaction with the system (4); the button panel offers the options of annotation, help, saving, logging, and exiting (5).

### A. SSW Network in the Case of Factory Automation

Due to the organization of the company development process, in the designed SSW network, the system workshops at the design level are grouped into two levels (see Fig. 13). At the upper sublevel, there is a system workshop (W-Company) that is devoted to the company d-experts, who are in charge of creating all virtual entities to be managed at the lower plane (including system workshops since they are complex virtual entities themselves); there is also a system workshop (W-HCI) that is devoted to HCI experts to check the created virtual entities. At the bottom sublevel, there is a system workshop (W-Personnel) to be used by the company d-experts that are in charge of generating the application workshops to be used by the company end users and a system workshop (W-Client) to be used by client company d-experts to create the application workshops devoted to their end users.

Such application workshops are related to two kinds of activities: 1) software mechanical design and testing of the automation systems; and 2) operation of the automation systems in the client factory. The workshops supporting activities of type 1 are devoted to different professionals working at the company, such as mechanical and software testers and mechanical programmers; the workshops supporting activities of type 2 are devoted to end users working in the client company, such as assembly line operators and production managers.

The application workshop devoted to the assembly line operators has been previously described in Fig. 12. W-Client is the system workshop permitting the mechanical engineers (client company d-experts) to create this application workshop. Both workshops are shown in gray in Fig. 13. W-Client permits the creation of the application workshop through simple drag-and-drop activities. Figs. 14 and 15 illustrate two different snapshots of this system workshop, generated during the interaction of a d-expert with the system workshop for creating the application workshop shown in Fig. 12. In Fig. 14, the virtual entity "canvas" has already been selected from the menu area on the right side and has been positioned on the work area to become the background of the workshop that is being created. The mechanical engineer is now dragging and dropping the virtual entity "bottoniera operativa" (operative button panel), which will appear on the canvas in its initial state, i.e., as a cp whose cs component is the displayed button panel, and the associated

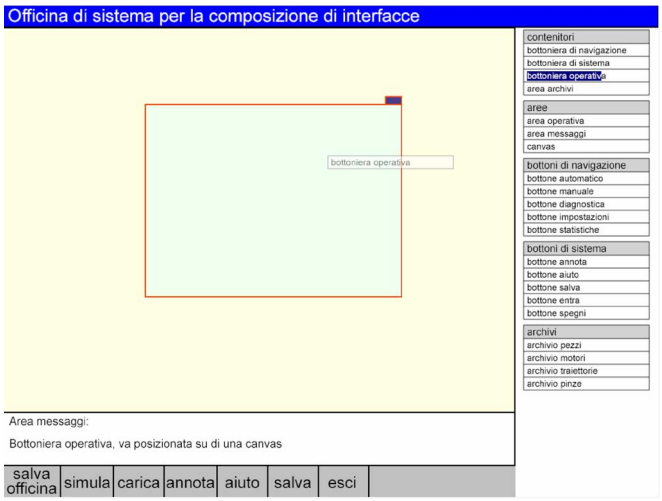



Fig. 14. Virtual entity "bottoniera operativa" (operative button panel) is dragged and dropped on the canvas representing the background of the workshop being created.


Fig. 15. Using W-Client system workshop, a d-expert is creating the application workshop shown in Fig. 12. The button "diagnostica" (diagnostic) is being located on the operative button panel.

program is able to manage the user interaction. This is an example of visual programming [18]. In Fig. 15, the application workshop presented in Fig. 12 is partially composed. The d-expert is now positioning a new button on the operative button panel.

### B. Communications in the Case of Factory Automation

In the network shown in Fig. 13, communications occur at the design level:

1) between the system workshops at the top plane, devoted to company d-experts and to HCI experts, respectively, who collaborate, interacting with their own system workshop, in developing the system workshop W-Personnel, which is devoted to the personnel working in their company, and the system workshop W-Client, which is devoted to the client company; they exchange the programs $P_{SSW}$ specifying the system workshops under development;
2) between a system workshop at the top plane and a system workshop at the bottom plane whenever W-Personnel and W-Client are generated or accessed from system workshops W-Company and W-HCI. Also, in this case, programs are exchanged.

At the use level, communications occur among the workshops devoted to the company personnel and among the workshops devoted to the company clients. In both cases, there is an exchange of data (results of data acquisition, computations, etc.) to test the final system (in the case of company personnel) and to use the interactive system to perform work tasks (in the case of the client companies).

Communications between different levels may occur: from the design level to the use level, application workshops are sent by company personnel or by the client company d-experts to end users (programs $P_{SSW}$ are exchanged); from the use level to the design level, annotations about usability problems or new user needs are sent, as in the medical case examined in Section V-D.

## VII. Related Work

The methodology described in this paper is the result of our initial experiences in the design of interactive systems, developed to support mechanical engineers, physicians, and photointerpreters. In developing the methodology, we followed a bottom-up iterative approach: from experiments [10] to the abstraction of a model [28], and again back to experiments, then to the revision of the model [8], [18], [19], [29]. In developing our experiments, we realized that, as software engineers, we know the technology; however, only domain experts understand the practice in the working environment.

Knowledge about user practice and future use situations can only be obtained by the collaboration with users, as agreed also by other researchers [30], [31]. Moreover, the increasing spread of computers in work environments determined the evolution of computer users from being software engineers or programmers to being just end users [8], [32].

The development of the methodology was influenced by the experiences of other groups, with similar goals. In the following sections, we briefly analyze those works that have been influential to our research or that, to the best of our knowledge, can be related to it.

### A. Model-Based Approaches

The SSW methodology is model-based in that it refers to a model of the interaction and coevolution processes of related members of a work domain to identify the causes of usability difficulties affecting interactive systems. The model is, thereafter, used to derive design procedures, which permit the implementation of systems in which these difficulties are eliminated or at least reduced. This model capitalizes on the seminal interaction model proposed by Hutchins *et al.* [33], which focuses on the human side of the interaction process and identifies the existence of semantic and articulatory

distances in evaluation and execution as the primary sources of usability difficulties. However, interaction processes are determined by a cognitive system (the human) and a computing system (the computer), which, in turn, form a unique system, called a "syndetic system" in [34], i.e., a system composed by binding subsystems of a different nature. To properly model the interaction process, one must also model the computing system, highlighting the problems arising on the computer side, i.e., capturing and interpreting the human actions. This stance is clearly posed in the model proposed in [35] and is also adopted by our model. However, we characterize each cycle of the interaction process as an exchange of messages that are subject to two interpretations: one that is performed by the user, and the second one that is internal to the system, associating the image on the screen with a computational meaning, as determined by the programs implemented in the system. The messages exchanged in the interaction process are seen as signs, which the humans must interpret within the context of their activity. This view is analogous to that independently proposed in computer semiotics [36], [37], a new discipline that "analyzes computer systems and their context of use under a specific perspective, namely as signs that users interpret to mean something" [38].

### B. Participatory Design Approaches

The SSW methodology is a participatory one, in that the design is performed by an interdisciplinary team, including representatives of end users, i.e., the domain experts [23].

Participatory approaches exploit techniques that are derived from social science that support communication and collaboration within the interdisciplinary team: these techniques move from system descriptions to collaborative construction of mockups, to cooperative prototyping and game-like design sessions. Most of these techniques not only analyze solutions after setting up goals but also use fantasy and imagined futures to study specific actions [39].

The Future Workshop technique foresees group meetings run by at least two facilitators to identify and analyze common problematic situations, generate visions about the future, and discuss how to realize these visions [40]. It is worth noting that in the Future Workshop technique, the word "workshop" denotes a seminar emphasizing exchange of ideas and practical methods. In our methodology, "workshop" denotes a small establishment where manufacturing or handicrafts are carried [41], as it is usual in the language of some professional people we collaborate with.

In cooperative prototyping [42], prototyping is viewed as a cooperative activity between users and designers. Prototypes are developed by software engineers, then are discussed with users, and are possibly experienced by them in work-like situations. Prototype modifications may be immediately made by direct manipulation, also by users, during each session of participatory design. However, in this approach, prototypes just represent an interactive digital evolution of paper-based mockups: real systems are then reprogrammed, and all modifications require a large programming effort that is made by designers after each session.

The SSW methodology favors the asynchronous exchange of ideas through the exchange of annotations and prototypes. It views prototyping as a cooperative incremental activity, in which all the stakeholders participate in the development of the final system. Each stakeholder operates on prototypes according to their own view, through the use of SSWs. Moreover, the use of prototypes permits the participation of d-experts and end users in the creation of software tools that they can tailor, customize, and program themselves in line with *participatory programming* [43]. Participatory programming is regarded as a way to transcend participatory design, but exploits traditional techniques of participatory design (*in situ* observations, interviews, workshops) to allow domain experts and software engineers to collaborate in developing and tailoring software tools.

On the whole, the SSW methodology favors the so-called "translation problem" among different stakeholders because it allows each stakeholder participating in the design process to interpret and experiment with the workshop being designed from her/his own point of view. This stance recalls the proposal in [44]. Moreover, we do not provide a mere translation, but present each stakeholder with a software environment supporting her/his own view of the task to be performed.

### C. EUD

The SSW methodology has been influenced by the work performed in EUD-Net, the network of Excellence on EUD, funded by the European Commission during 2002 and 2003 [6]. In the literature, end-user programming and end-user computing are often used as interchangeable terms; for example, in [45], the authors discuss "enhancing editors with *end-user programming capabilities*." They also say that "end-user computing is needed in domains or applications where the activity cannot be planned in advance," and that it should have the flavor of "on-the-fly" computing, i.e., it should emerge during the user activity, when the user needs to create a combination/repetition/abstraction construct, according to some concrete situation. Brancheau and Brown [12] describe *end-user computing* as "the adoption and use of information technology by people outside the information system department, to develop software applications in support of organizational tasks." The EUD-Net collaboration preferred the term EUD to indicate the active participation of end users in the software development process; this can range from providing information about requirements, use cases, and tasks, including participatory design, to activities such as customization, tailoring, and coevolution.

A system acceptable by its users should have a gentle slope of complexity: this means that it should avoid big steps in complexity and keep a reasonable tradeoff between ease-of-use and functional complexity. For example, systems might offer end users different levels of complexities in performing EUD activities, going from simply setting parameters, to integrating existing components, up to extending the system by developing new components [46], [47]. In this line, Mørch [48] proposes three levels of tailoring: customization, integration, and extension. *Customization* usually consists of configuring a set of

preferences performed by the user through a preference form by setting parameters for the various configuration options the application supports. *Integration* goes beyond customization and allows users to add new functionalities to an application by linking together predefined components without accessing the underlying implementation code. *Extension* refers to the case in which the application does not provide, by itself or by its components, any functionality that accomplishes a specific user need; thus, adding a new functionality generates a radical change in the software. The SSW methodology encompasses all these levels of tailoring. It also takes into consideration the results in [22] and [49], where empirical studies are reported on the activities end users are willing to perform. Mackay [22] analyzes how users of a UNIX software environment try to customize the system. She finds that many end users do not customize their applications as much as they could. This depends on the fact that it takes too much time and deviates from other activities. Nardi [49] conducted empirical studies on the users of spreadsheets and CAD software. She found that those users perform activities that generate new software artifacts and are even able to master the formal languages embedded in these applications when they have a real motivation for doing so.

Software technology has advanced to the point that we can build tools for end users to design systems by interacting with icons and menus in graphical microworlds. Several researchers working on EUD capitalize on this and describe technologies for component-based design environments (e.g., [50]), libraries of patterns, and templates. There are various proposed design environments that do not require users to program *per se*; instead, they design by instructing the machine to learn from examples [51]. In this line, system workshops devoted to d-experts permit creating programs just by visually composing virtual entities selected from repositories, as described in the two case studies.

Giving end users ways to easily create their own programs is important; however, it is not enough. Like their counterparts in the world of professional software development, end users need support for other aspects of the software life cycle. In particular, referring to the case of errors in end-user programs, such as formula errors in spreadsheets, Burnett *et al.* [52] discuss a strategy that gives end users the ability to perform quality-control methods. Myers *et al.* [53] work to develop natural programming languages and environments to permit people to program by expressing their ideas in the same way that they think about them.

### D. Metadesign

EUD can be considered a two-phase process: the first phase being designing the design environment, and the second one being designing the applications using the design environment. These two phases are not clearly separated 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 comments of end users at work.

Our approach is consistent with this concept of metadesign, which is explored in [54]. Metadesign is a process in which end users are able to act as designers and contribute to the coevolution of the system. Metadesign must enable humans to shape their sociotechnical environments and empower them to adapt their tools to their own needs.

In this perspective, metadesign highlights the novel vision of interactive systems that is adopted by the SSW methodology. Specifically, it emphasizes that all the stakeholders involved in the design of the system are "owners" of a part of the problem [54], and, therefore, each is provided with an SSW, through which the stakeholder can contribute to shape software artifacts. HCI experts, software engineers, and end users acting as developers, each one through her/his SSW, can access and modify the system of interest in accordance with her/his own culture, experience, needs, and skills. They can also exchange among themselves and evaluate the results of these activities to converge on a common design. In light of these considerations, we view metadesign as *a design paradigm that includes end users as active members of the design team and provides all the stakeholders in the team with suitable languages and tools to foster their personal and common reasoning about the development of interactive software systems that support the end users' work.*

### E. User Diversity

The SSW methodology emphasizes the need of developing different software environments for end users working in the same domain but with different roles. Similarly, the Design Aid for Intelligent Support System (DAISY) is a design methodology for building decision support systems in complex experience-centered domains [55]. It provides a technique for identifying the specialized needs of end users within a specific range of domain experience. In other domains, systems can have multiple end users with multiple roles. As an example, the Dynamic Interaction Generation for Building Environments (DIGBE) is a system that creates end-user interfaces adapted to the multiple end users with different roles that collaborate to the management of a building control system [56]. An expert user, typically a building manager, sets the initial state of the systems dedicated to other expert users (i.e., managers, operators, and technicians). The generated prototypes are similar to the prototypes developed in recent years through the SSW methodology in the mechanical engineering case study described in Section VI.

The SSW methodology emphasizes the importance of both context of activity and working organization and, as suggested by activity theory [57], considers software systems as artifacts having a mediating role between objects and subjects of activities. In fact, end users work in a context outside the computer and are required to apply their intuitive knowing to reflect on the current situation and decide what to do next. Following Schön [13], we assume that end users perform their activity as competent practitioners. The interactive system should support end users in exploiting their practical competence and skill.

## VIII. Conclusion

We have presented the SSW methodology that is aimed at designing software systems that support end users to tailor

and even design their tools, i.e., to perform EUD activities. Insights emerging from our recent experiences have been provided.

This paper contributes to the research on EUD by: 1) founding EUD on a model of the interaction and coevolution processes; 2) providing the *SSW design methodology*, based on that model, to build software environments (SSWs) that are customized to the needs of the end-user communities to which they are devoted and that allow them to perform EUD activities. The SSWs also provide tools that support the communication among d-experts, HCI experts, and software engineers and their collaborative participation in the design process.

The SSW approach improves the interaction between users and systems by providing advantages from different points of view.

1) A pragmatic point of view: the approach permits a progressive refinement of the specification of the interaction process defined from the point of view of the software engineers, at the top level, into a concrete specification of the same process in terms of the notations familiar to the end users (e.g., mechanical engineers, medical doctors), thus supporting a gentle slope of complexity [46], [47].
2) A semantic point of view: the approach favors the match between the computational meaning of the software tools and the meaning usually given to them by the different users (HCI designers, domain experts, and end users). At each level of the hierarchy, every SSW adopts an interaction visual language [19] based on the notations of the users working with it. The SSWs present domain experts with virtual entities whose characteristic structures and computational constructs resemble those of the entities experts use in their traditional working environment.
3) A communication point of view: the approach permits message exchange among the different stakeholders through the SSW network. Messages are presented to the various communicants in their contexts of activity and according to their interpretation schemata.

End users involved in the two described projects are collaborating with enthusiasm to the development of the SSW prototypes. They understand and appreciate the novel approach of being involved in collaborative design processes, through which they can have a more active role than simple consumers of new technologies. The possibility of having software environments that they can easily adapt to their needs is a new perspective that excites them very much. Moreover, SSWs permit end users to carry out some activities much faster than the conventional face-to-face methods that usually require a long time (e.g., exchanging consultations). The comments we collected after observing people using the developed prototypes are a clear sign that this approach is well accepted by users and generates more pleasure and fun in their overall experience with new technology.

## ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their useful comments, and also G. Fresta for the stimulating discussions during the development of this work and for his contribution to the implementation of the prototypes presented in this paper. The authors are grateful to ETA Consulting in Brescia (and particularly Silvano Biazzi) and to the physicians of the Hospital "Giovanni XXIII" in Bari for their collaboration in the case studies.

## REFERENCES

[1] B. Shneiderman, *Leonardo's Laptop: Human Needs and the New Computing Technologies*. Cambridge, MA: MIT Press, 2002.
[2] G. Fischer, "Beyond 'couch potatoes': From consumers to designers and active contributors," *FirstMonday*. [Online]. Available: http://firstmonday.org/issues/issue7_12/fischer/
[3] M. Davis, "Reenvisioning visual information systems: From signal analysis to context-aware media computing," in *Proc. Int. Conf. DMS*, San Francisco, CA, 2004. Keynote Speech.
[4] J. Nielsen, *Usability Engineering*. San Diego, CA: Academic, 1993.
[5] J. M. Carroll and M. B. Rosson, "Deliberated evolution: Stalking the view matcher in design space," *Hum.-Comput. Interact.*, vol. 6, no. 3/4, pp. 281–318, 1992.
[6] *EUD-Net Thematic Network*. [Online]. Available: http://giove.cnuce.cnr.it/eud-net.htm
[7] A. Sutcliffe and N. Mehandjiev, "End-user development," *Commun. ACM*, vol. 47, no. 9, pp. 31–32, Sep. 2004.
[8] M. F. Costabile, D. Fogli, P. Mussio, and A. Piccinno, "End-user development: The software shaping workshop approach," in *End-User Development*, H. Lieberman, F. Paternò, and V. Wulf, Eds. Dordrecht, The Netherlands: Kluwer, 2006, pp. 183–205.
[9] A. Cypher, *Watch What I Do: Programming by Demonstration*. Cambridge, MA: MIT Press, 1993.
[10] P. Mussio, M. Pietrogrande, and M. Protti, "Simulation of hepatological models: A study in visual interactive exploration of scientific problems," *J. Vis. Lang. Comput.*, vol. 2, no. 1, pp. 75–95, 1991.
[11] P. Carrara, D. Fogli, G. Fresta, and P. Mussio, "Toward overcoming culture, skill and situation hurdles in human–computer interaction," *Int. J. Universal Access Inf. Soc.*, vol. 1, no. 4, pp. 288–304, Aug. 2002.
[12] J. C. Brancheau and C. V. Brown, "The management of end-user computing: Status and directions," *ACM Comput. Surveys*, vol. 25, no. 4, pp. 437–482, Dec. 1993.
[13] D. Schön, *The Reflective Practitioner—How Professionals Think in Action*. New York: Basic Books, 1983.
[14] T. R. G. Green and M. Petre, "Usability analysis of visual programming environments: A 'cognitive dimensions' framework," *J. Vis. Lang. Comput.*, vol. 7, no. 2, pp. 131–174, Jun. 1996.
[15] A. Dillon and C. Watson, "User analysis in HCI: The historical lesson from individual differences research," *Int. J. Hum.-Comput. Stud.*, vol. 45, no. 6, pp. 619–637, 1996.
[16] J. R. Hayes, *Three Problems in Teaching General Skills*, vol. 2. Hillsdale, NJ: Lawrence Erlbaum, 1985.
[17] A. Dix, J. Finlay, G. Abowd, and R. Beale, *Human Computer Interaction*. London, U. K.: Prentice-Hall, 2004.
[18] P. Bottoni, M. F. Costabile, S. Levialdi, and P. Mussio, "Defining visual languages for interactive computing," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 27, no. 6, pp. 773–783, Nov. 1997.
[19] P. Bottoni, S. K. Chang, M. F. Costabile, S. Levialdi, and P. Mussio, "Modelling visual interactive systems through dynamic visual languages," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 32, no. 6, pp. 654–669, Nov. 2002.
[20] G. Bourguin, A. Derycke, and J. C. Tarby, "Beyond the interface: Co-evolution inside interactive systems—A proposal founded on activity theory," in *Proc. IHM-HCI*, Lille, France, 2001, pp. 297–310.
[21] S. Arondi, P. Baroni, D. Fogli, and P. Mussio, "Supporting co-evolution of users and systems by the recognition of interaction patterns," in *Proc. Int. Conf. AVI*, Trento, Italy, 2002, pp. 177–189.
[22] W. E. Mackay, "Triggers and barriers to customizing software," in *Proc. CHI Human Factors Comput. Syst.*, New Orleans, LA, 1991, pp. 153–160.
[23] D. Schuler and A. Namioka, "Preface," in *Participatory Design, Principles and Practice*, D. Schuler and A. Namioka, Eds. Hillsdale, NJ: Lawrence Erlbaum, 1993, pp. xi–xiii.

[24] H. Rittel, "Second-generation design methods," in *Developments in Design Methodology*, N. Cross, Ed. New York: Wiley, 1984, pp. 317–327.
[25] J. Preece, *Human–Computer Interaction*. Boston, MA: Addison-Wesley, 1994.
[26] D. Fogli, G. Fresta, A. Marcante, and P. Mussio, "Two-way exchange of knowledge through visual annotation," in *Proc. DMS*, San Francisco, CA, 2004, pp. 286–291.
[27] H. R. Hartson and D. Hix, *Developing User Interfaces*. New York: Wiley, 1993.
[28] P. Mussio, M. Finadri, P. Gentini, and F. Colombo, "A bootstrap approach to visual user-interface design and development," *Vis. Comput.*, vol. 8, no. 2, pp. 75–93, Feb. 1992.
[29] S. K. Chang and P. Mussio, "Customized visual language design," in *Proc. Int. Conf. SEKE*, 1996, pp. 553–562.
[30] G. Fischer, "Seeding, evolutionary growth, and reseeding: Constructing, capturing, and evolving knowledge in domain-oriented design environments," *Autom. Softw. Eng.*, vol. 5, no. 4, pp. 447–468, 1998.
[31] K. Grønbæk, J. Grudin, S. Bødker, and L. Bannon, "Achieving cooperative system design: Shifting from a product to a process focus," in *Participatory Design, Principles and Practice*, D. Schuler and A. Namioka, Eds. Hillsdale, NJ: Lawrence Erlbaum, 1993, pp. 79–97.
[32] J. Grudin, "The computer reaches out: The historical continuity of interface design," in *Proc. CHI Conf.*, Seattle, WA, 1990, pp. 261–268.
[33] E. L. Hutchins, J. D. Hollan, and D. Norman, "Direct manipulation interfaces," in *User Centred System Design*, D. Norman and S. Draper, Eds. Hillsdale, NJ: Lawrence Erlbaum, 1986, pp. 87–124.
[34] P. Barnard, J. May, D. Duke, and D. Duce, "Systems, interactions, and macrotheory," *ACM Trans. Comput.-Hum. Interact.*, vol. 7, no. 2, pp. 222–262, 2000.
[35] G. D. Abowd and R. Beale, "Users, systems and interfaces: A unifying framework for interaction," in *Proc. HCI: People and Comput. VI,* D. Diaper and N. Hammon, Eds., 1991, pp. 73–87.
[36] P. B. Andersen, "What semiotics can and cannot do for HCI," *Knowl.-Based Syst.*, vol. 14, no. 8, pp. 419–424, Dec. 2001.
[37] C. S. de Souza, *The Semiotic Engineering of Human Computer Interaction*. Cambridge, MA: MIT Press, 2005.
[38] P. B. Andersen, "Computer semiotics," *Scand. J. Inf. Syst.*, vol. 4, pp. 3–30, 1992.
[39] S. Bødker, K. Grønbæk, and M. Kying, "Cooperative design: Techniques and experiences from the Scandinavian scene," in *Participatory Design—Principles and Practices*, D. Schuler and A. Namioka, Eds. Hillsdale, NJ: Lawrence Erlbaum, 1993, pp. 157–175.
[40] J. Greenbaum and M. Kyng, Eds., *Design at Work: Cooperative Design of Computer Systems*. Hillsdale NJ: Lawrence Erlbaum, 1991.
[41] *Merriam-Webster online*. [Online]. Available: http://www.webster.com
[42] S. Bødker and K. Grønbæk, "Design in action: From prototyping by demonstration to cooperative prototyping," in *Design at Work: Cooperative Design of Computer Systems*, J. Greenbaum and M. Kyng, Eds. Hillsdale, NJ: Lawrence Erlbaum, 1991, pp. 197–218.
[43] C. Letondal and W. E. Mackay, "Participatory programming and the scope of mutual responsibility: Balancing scientific, design and software commitment," in *Proc. PDC*, Toronto, ON, Canada, 2004, pp. 31–41.
[44] R. DePaula, "Lost in translation: A critical analysis of actors, artifacts, agendas, and arenas in participatory design," in *Proc. PDC*, Toronto, ON, Canada, 2004, pp. 162–172.
[45] M. Balaban, E. Barzilay, and M. Elhadad, "Abstraction as a means for end-user computing in creative applications," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 32, no. 6, pp. 640–653, Nov. 2002.
[46] B. A. Myers, D. C. Smith, and B. Horn, "Report of the 'end-user programming' working group, languages for developing user interfaces," Jones & Bartlett, Boston, MA, pp. 343–366, 1992.
[47] V. Wulf and B. Golombek, "Direct activation: A concept to encourage tailoring activities," *Behav. Inf. Technol.*, vol. 20, no. 4, pp. 249–263, Jul. 2001.
[48] A. Mørch, "Three levels of end-user tailoring: Customization, integration, and extension," in *Computers and Design in Context*, M. Kyng and L. Mathiassen, Eds. Cambridge, MA: MIT Press, 1997, pp. 51–76.
[49] B. Nardi, *A Small Matter of Programming: Perspectives on End User Computing*. Cambridge, MA: MIT Press, 1993.
[50] A. I. Mørch, G. Stevens, M. Won, M. Klann, Y. Dittrich, and V. Wulf, "Component-based technologies for end-user development," *Commun. ACM*, vol. 47, no. 9, pp. 59–62, Sep. 2004.
[51] H. Lieberman, *Your Wish is My Command: Programming by Example*. San Francisco, CA: Morgan Kaufman, 2001.
[52] M. Burnett, C. Cook, and G. Rothermel, "End-user software engineering," *Commun. ACM*, vol. 47, no. 9, pp. 53–58, Sep. 2004.
[53] B. A. Myers, J. F. Pane, and A. Ko, "Natural programming languages and environments," *Commun. ACM*, vol. 47, no. 9, pp. 47–52, Sep. 2004.
[54] G. Fischer, E. Giaccardi, Y. Ye, A. G. Sutcliffe, and N. Mehandjiev, "Meta-design: A manifesto for end-user development," *Commun. ACM*, vol. 47, no. 9, pp. 33–37, Sep. 2004.
[55] C. B. Brodie and C. C. Hayes, "DAISY: A decision support design methodology for complex, experience-centered domains," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 32, no. 1, pp. 50–71, Jan. 2002.
[56] R. R. Penner and E. S. Steinmetz, "Model-based automation of the design of user interfaces to digital control systems," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 32, no. 1, pp. 41–49, Jan. 2002.
[57] K. Kuutti, "Activity theory as a potential framework for human–computer interaction," in *Context and Consciousness: Activity Theory and Human Computer Interaction*, B. Nardi, Ed. Cambridge, MA: MIT Press, 1996, pp. 17–44.

**Maria Francesca Costabile** (M'87–SM'04) received the Laurea degree in mathematics from the University of Calabria, Cosenza, Italy.

From 1981 to 1988, she was with the University of Calabria as an Assistant Professor. From 1989 to 1999, she was with the University of Bari, Bari, Italy, as an Associate Professor. Since year 2000, she has been a Full Professor with the Dipartimento di Informatica, Università di Bari, where she teaches HCI and other courses for the computer science curriculum, of which she is currently the coordinator. She has been a Visiting Scientist in several foreign universities, primarily in the U.S. and Germany. She has published over 150 papers in scientific journals, books, and proceedings of international conferences, and edited six books published by ACM Press and Springer. Her current research interests are in HCI, visual system design, multimodal and multimedia interaction, usability engineering, adaptive interfaces, user models, end-user development, and information visualization.

Prof. Costabile is regularly in the program committees of international conferences and workshops. She was the Program Chair of Advanced Visual Interfaces 2004 and a Program Co-Chair of Interact 2005. She is a Program Co-Chair of CHI 2008. She is a member of Association for Computing Machinery (ACM) and the ACM Special Interest Group on Computer–Human Interaction (SIGCHI). She is a Founding Member of the Italian Chapter of ACM SIGCHI and served as Chair from 1996 to 2000.

**Daniela Fogli** (M'04) received the Laurea degree in computer science from the University of Bologna, Bologna, Italy, in 1994 and the Ph.D. degree in information engineering from the University of Brescia, Brescia, Italy, in 1998.

From 1998 to 2000, she was a Ph.D. grant holder at the Joint Research Centre of the European Commission, Institute for Systems, Informatics and Safety. Since 2000, she has been an Assistant Professor with the Dipartimento di Elettronica per l'Automazione, Università di Brescia. In 2005, she has been a Visiting Scholar with the Center for Life-Long Learning and Design, University of Colorado, Boulder. She has published more than 60 scientific papers. Her current research interests include specification and design of visual interactive systems, Web technologies supporting the implementation of visual interactive systems, metadesign, and end-user development.

Dr. Fogli served as short papers Co-Chair of the Association for Computing Machinery (ACM) Conference on Advanced Visual Interfaces (AVI'06). She is a member of the ACM and Special Interest Group on Computer–Human Interaction (SIGCHI) Italy (the Italian Chapter of ACM SIGCHI).

**Piero Mussio** received the Laurea degree in electronic engineering from the Politecnico di Milano, Milano, Italy. He was a Researcher at the Laboratory of Cosmic Physics, National Research Council of Italy (LFCTR), Milan. At the LFCTR, he was a Data Reduction Officer in European collaborations and in charge of the Group of Data Analysis. In 1983, he joined the Department of Physics, Milan University, Milan, as an Associate Professor of computer science, where he was responsible for the Image Processing and Interpretation Group. From 1996 to 2003, he was with the University of Brescia, Brescia, Italy, as a Professor in computer science. He is currently a Full Professor with the Dipartimento di Informatica e Comunicazione, Università di Milano. His research interests evolved from pattern recognition in images and complex system behaviors to pattern recognition in HCI, visual computing, and in visual language, and visual computing environment design, formal specification, and experimental validation. He is an Associate Editor of the *Journal of Visual Languages and Computing*.

Prof. Mussio served as the President of the Italian Chapter of the International Association for Pattern Recognition (IAPR). He served as the Program Chair for the 1999 IEEE Workshop on Visual Languages, Tokyo, 1999; General Co-Chair of 2000 IEEE Workshop on VL, Seattle, 2000; and Program Co-Chair of Association for Computing Machinery (ACM) Conference on Advanced Visual Interfaces (AVI'06), Venezia, 2006. He is an IAPR Fellow, a member of ACM, and a member of the Special Interest Group on Computer–Human Interaction (SIGCHI) Italy (the Italian Chapter of ACM SIGCHI). He was a Program Committee Member for conferences, schools, and workshops on visual languages, pattern recognition, machine vision, and HCI.

**Antonio Piccinno** received the Laurea degree in computer science with full marks and honors and the Ph.D. degree in computer science from the University of Bari, Bari, Italy, in 2001 and 2005, respectively.

Since April 2005, he has been an Assistant Researcher with the Dipartimento di Informatica, Università di Bari. His research interests focus on HCI, visual interactive systems, theory of visual languages, end-user development, metadesign, component-based software development, WWW Interfaces, multimodal and multimedia interaction, and adaptive interfaces.

Dr. Piccinno served in the scientific secretariat of the International Conference Advanced Visual Interfaces (AVI), 2004. He is a member of ACM, ACM Special Interest Group on Computer–Human Interaction (SIGCHI), and SIGCHI Italy (the Italian Chapter of ACM SIGCHI).