An agent-based architecture to support knowledge management in interactive system life-cycle

Pietro Baroni, Daniela Fogli, Piero Mussio

Abstract— **Several knowledge management issues arise during the life-cycle of an interactive system. In the design phase software engineers need to elicit from users and domain experts knowledge about what they need and about the intended use of the interactive system. Interacting with the system, users modify their needs and/or their ways of using the system: knowledge about new user habits becomes then a key factor for user-system co-evolution, i.e. for useful software maintenance and extension interventions. In this paper we propose an agent-based architecture to support knowledge management in the complex process of user and system co-evolution. The architecture is based on the visual workshop hierarchy approach, where different kinds of interactive systems are associated in a hierarchical fashion with the different figures involved in system life-cycle. The approach is complemented with a community of agents which may play two kinds of roles: observer agents in charge of observing and recording user-system interaction at each level of the hierarchy, and recognition agents in charge of analyzing observation records and of extracting patterns of interactions to be submitted to the attention of designers at the appropriate level. Examples from a case study make discussion on the proposed architecture concrete and a sketch about the implementation is provided.**

Index Terms—**Interactive system design, System observation, Coevolution, Knowledge management.**

I. Introduction

Several approaches to the design of Interactive Systems (ISs) ascribe a key role to the end-users. For instance, both the *participatory* [1] and the *user centered* [2] approaches assume that the user experience drives the design. More recently, some researches propose *collaborative* approaches, in which the design is performed by an interdisciplinary team, including some representative of users, called *domain experts* [3]. As a matter of fact, these and similar approaches are based on the simple consideration that the most important knowledge for IS design, i.e. *knowledge* about what users do and need, is possessed by the users themselves.

P. Baroni, D. Fogli and P. Mussio are with the Dipartimento di Elettronica per l'Automazione, Università di Brescia, Via Branze 38, 25123 Brescia, Italy (corresponding author: D. Fogli, phone: +39-030-3715455; fax: +39-030-380014; e-mail: {baroni, fogli, mussio}@ ing.unibs.it).

This work is supported by Italian Ministry of University and Research, MIUR, in the framework of the project *Specification, Design and Development of Visual Interactive Systems - PRIN 2000* and by ex60% funding programme

Adopting this standpoint, some well-known problems in the area of IS design can be regarded as typical knowledge management problems.

In particular, a pathological phenomenon, which often affects the design and implementation process of Interactive Systems and leads to the development of ISs difficult to learn and use, is the *communicational gap* existing between users and software designers [4][5].

This phenomenon is related with the variety and complexity of the knowledge involved in IS design, which pose a serious problem of knowledge elicitation and sharing.

In fact, at least three kinds of professional figures are typically involved in the development process of an IS: software engineers, domain experts, and end users.

The communicational gap arises from the fact that these professional figures correspond to different cultural backgrounds, and, as a consequence, detain distinct types of knowledge and follow different approaches and reasoning strategies to modeling, performing and documenting the tasks to be achieved in a given application domain.

In particular, one of the main causes of the communicational gap is the different level of abstraction adopted by different professional figures in their reasoning and language. Moreover, professionals with different cultural backgrounds may adopt different approaches to abstraction, since, for instance domain experts and software designers may have different notions about the details that can be abstracted away.

Users reason heuristically rather than algorithmically, using examples and analogies rather than deductive abstract tools, documenting activities, prescriptions and results, through their own developed notations. These notations are not defined according to computer science formalisms but they are concrete and situated in the specific context, in that they are based on icons, symbols and words that resembles and schematise the tools and the entities which are to be operated in the working environment. They emerge from users' practical experiences in their specific domain of activity [4][6]. These notations highlight those kinds of information users consider important for achieving their tasks, at the expense of obscuring other kinds [7], and facilitate the heuristic problem solving strategies, adopted in the specific user community.

Moreover specialized *user dialects* stem from *user diversity*

[3], rising from the existence of users sub-communities which develop peculiar abilities, knowledge and notations for example for the execution of specialized subtasks.

Another relevant problem, which has received less attention in the Human Computer Interaction (HCI) literature, but that is well known in the knowledge acquisition field, is the existence of *implicit knowledge,* namely the knowledge that a person possesses and currently uses to carry out tasks and solve problems but that s/he is unable to express in verbal terms and that s/he may even be unaware of.

Also in this case, it is a common experience that in many application fields end-users possess a large amount of implicit knowledge, since they are often more able to do than to explain what they do. While this may, in general, represent a very difficult problem, in the area of IS design its solution can be favored by the fact that the ways users understand and exploit a system have, in any case, a precise and observable counterpart in the actions they execute while interacting with it. Therefore one way to make this implicit knowledge explicit consists in observing end-users during their interaction with an existing system or, if the system has not been developed yet, by observing interactions of users with system mock-ups. Mock-ups, often prepared in the early phases of system projects, are used to have a first feedback from users with respect to the system being developed, in order to better understand their needs and expectations.

User observation techniques are discussed for instance in [8]. Some of these techniques are based on the presence of a human observer and suffer therefore by two main kinds of limitations:

- on one side, human observers have limited observation capabilities: some relevant aspects of interaction may escape their attention; moreover they can be biased in collecting use data, especially if they coincide with software designers, while, on the other hand, their presence may influence the users and induce unnatural behaviors [9];
- on the other side, they are generally applied only in the design phase of IS life-cycle: once the IS has been released, user observation ceases.

The latter kind of limitation, is particularly significant from the viewpoint of a correct knowledge management along the IS life-cycle. In fact, even if it would be possible to correctly elicit all end-users knowledge during system development or by observing the users interacting with the IS in the early stages of use, this knowledge will change and increase due simply to the fact that users use the system. In fact, an important 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" [10].

In turn, the designer evolves the system to adapt it to its new usages. We call this phenomenon *co-evolution of users and systems*, to emphasize the interest on methods and tools to support adequate system co-evolution [5].

Co-evolution stems from two main sources: a) user creativity: the users may devise novel ways to exploit the system in order to satisfy some needs not considered in the specification and design phase; and b) user acquired habits: 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 non calculation data in spreadsheets, which was included in later versions of spreadsheets, after the observation that users frequently forced the spreadsheet to manage non-calculation data for data archiving and other tasks [11].

An example of co-evolution 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 are required for 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 have been co-evolved to encompass this user behavior: when a new directory is created in the "Save as" context, it automatically becomes the current one.

In order to overcome the communicational gap, to manage user diversity and the implicit knowledge problem, representatives of the user community are recruited in the design team as *domain experts*. They are experienced users, who possess a higher level and more complete view of the application domain, with the capability of distinguishing and characterizing user groups and the relevant dialects. However they may lack knowledge about some operational details which are critical for the final acceptance and usability of a IS by end users and may not be aware of all existing dialects and their features. These dialects remain *unknown* during the first design phase.

Domain expert complement software designers, who have limited or even no domain knowledge at all. Software engineers contribute to the design process their competence in formal modeling and software engineering principles and techniques. However, software engineers tend to reason about the problems in an abstract way, because they focus on abstraction and generalization of models, algorithmic description of activities, formal verification, software reuse and maintenance. For this reason, designers often tend to pay little attention to the mapping between programs and real cognitive processes and to the ease of user interactive solution of problems. The result of these design approaches are ISs which users access and steer only being forced to express their problems in the computer oriented notations imposed them by the designers. These ISs impose their grain to users resolution strategies, a grain often not amenable to user reasoning, and possibly even misleading for them [6].

The need for managing co-evolution and the existence of unknown dialects call for the design of systems which support users and designers in identifying the improvements to be

performed and can be incrementally adapted to these findings emerging from its use in practice.

To reach these goals, this paper proposes an agent-based architecture, capitalizing on openness of multi-agent architectures, and agent pro-activeness, autonomy, social ability and reactivity.

The rest of the paper is organized as follows. In Section 2 we present a case study which exemplifies the concepts and problems introduced above. In Section 3 we describe the visual workshop hierarchy approach to design of Visual Interactive Systems (VISs), namely interactive systems based on visual interaction, while in section 4 we present a prototypal system for user observation and extraction of recurrent patterns of interactions. In section 5 we show how these concepts can be combined within an agent-based architecture supporting the management of knowledge concerning IS design and co-evolution. In section 6 a discussion about the peculiarities of our work and a comparison with recent literature are provided. Finally in section 7 we sketch future work directions and conclude the paper.

II. KNOWLEDGE MANAGEMENT WITHIN A WORKING ORGANIZATION: A CASE STUDY

To make the above discussion concrete we introduce an example taken from a case - studied in [12] and [13] - in which Earth scientists and technicians analyze satellite glacier images to obtain medium and long term environmental forecast and organize the forecast results into reports and thematic maps for different communities of client experts (planners, decision makers, ...). Reports may include photographs, graphs, etc., and textual or numeric data related to the environmental phenomena of interest.

The team of designers, including software engineers, domain experts and HCI experts, analysed the tasks performed in a specific working environment, for which a VIS has to be developed. The team recognized two kinds of activity: photo-interpreters classify, interpret and annotate remote sensed data of glaciers; clerks organize the interpreted images into documents to be delivered to different communities of clients.

Photo-interpreters and clerks represent two sub-communities of end-users within the Earth Scientist & Technologist community: they share environmental data archives, some models for their interpretation, some notations for their presentation but have also to achieve different tasks, documented through different sub-notations and tools. Therefore, their notations can be considered two (visual) dialects of the Earth Scientist & Technologist general notation.

The team of designers decided to develop two separate but consistent environments, called *application workshops* in our terminology [13]. The first, *B-glacier*, was developed for the photo-interpreter community, equipped with tools for interactive image processing and data annotation and for archiving the interpreted images and annotations. The second, *B-monitore*, was developed for the clerk community, equipped with tools for the retrieval of data and images, for their annotation, for organizing images, sketches, graphs and texts into documents, for archiving and dispatching the produced documents. Moreover, the team of designers also observed that adaptation of *B-glacier* and *B-monitore* to different tasks and situations requires the knowledge of both dialects and activities, of the tasks to be executed and of the working organization, and the awareness of the use of the reports outside the organization. Only senior Earth scientists may have such a knowledge. Therefore, the team decided that a senior expert should act as a manager of the whole activity and be responsible of recognizing the tasks to be performed, identifying the dialect notations of interest, and consequently defining the whole VIS as a system of consistent application workshops. The senior scientist achieves these goals using another environment, a *system workshop*, called *B-GlacManager*, where s/he finds usable tools for implementing and adapting application workshops (see fig. 1).

Several co-evolution phenomena may concurrently arise in such a context. End users may change the way of using their application workshops: if the senior scientist is able to correctly capture these changes, s/he can introduce suitable modifications and extensions in the application workshops using the system workshop. However if the requirements imposed by the evolution go beyond the capabilities initially included in the system workshop, the senior scientist has to resort to the help of software designers.

In turn, while producing and/or modifying application workshops, the senior scientist may change his/her way of using the system workshop: this phenomenon should be captured by software engineers to evolve the system workshop, with potential indirect benefits on the application workshops of end-users.

Correctly supporting co-evolution through proper knowledge extraction and management may significantly improve the usability, or even the acceptability, of the workshops at different levels with a remarkable impact on the productivity of the professionals involved and on the quality of the results of their work.

III. THE VISUAL WORKSHOP HIERARCHY APPROACH

In the visual workshop hierarchy approach [13], each VIS is designed as it would be a *virtual workshop*, i.e. an environment in which users find and use virtual tools familiar to them and used in their everyday activities according to their habits. In fact, domain experts and end-users know the real tools they are familiar with and their own habits, so this knowledge can be easily applied to similar virtual tools. On the other hand, software engineers know abstraction techniques, programming tools and their grains and exploit this knowledge in the system development.

The visual workshop hierarchy strategy is therefore a collaborative approach which capitalizes on the different knowledge sources of the professional figures involved in the use and development of an IS. In particular, end-users, domain

experts and software engineers cooperates to identify 'what' and 'how' to do, namely to identify the application workshops to be developed. The team of designers develop their own ISs (system workshops) and use them to collaborate in the development of application workshops. System and application workshops form a hierarchy which arises from the working organization of the user community. Figure 1 shows the hierarchy in the case of Earth Science.

In general, the hierarchy organization depends on the working organization of the end-user community: the designer team organizes each hierarchy into a number of levels and, at each level, defines a number of workshops, depending on this organization. Each workshop is devoted to the execution of tasks of a same type. The top level (software engineering level), and the bottom level (application level), are always present in a hierarchy.

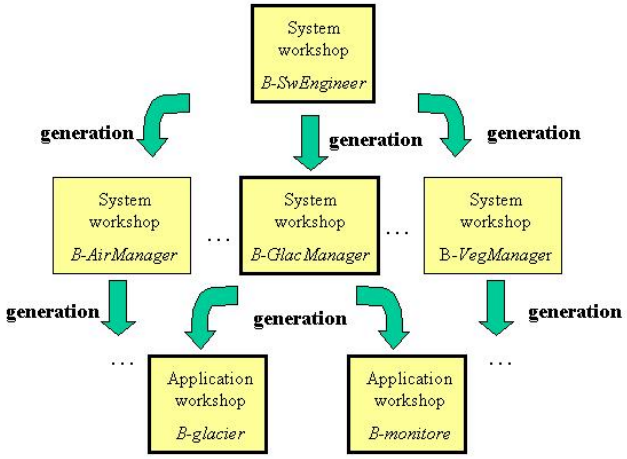

Figure 1. A 3-level visual workshop hierarchy.

The hierarchy should be designed so that, in each workshop, users find all but only the tools required to perform the specific task type, according to their culture, skill and experience. During task execution, at each stage of the work, the system presents its users a working area, called 'bench'. The user selects from a tool repository and makes available on the bench the tools required in that work stage. The user selects from data repositories the entities to be worked on, and performs the task applying the tools on these entities. In this way, users should more easily orient their navigation in the virtual space to achieve their goals and avoid to loose themselves in the virtual space. However, in our experience, all working situations cannot be foreseen in advance. Therefore, in each workshop, it must be possible for the user to perform some local adaptation.

The visual workshop hierarchy approach acknowledges the existence of different types of knowledge corresponding to different professional figures and aims at favoring the evolution towards providing the most appropriate tool to each figure. To fulfill this goal, it has to include automated observation and knowledge extraction techniques supporting co-evolution.

IV. SUPPORTING CO-EVOLUTION OF INTERACTIVE SYSTEMS

Co-evolution is a word widely used in scientific works, from Carroll and Rosson's co-evolution of users and tasks [14] to the co-evolution of artifacts supporting HCI design in the different steps of the product lifecycle, with the aim of obtaining a consistent set of tools [15]. Co-evolution of users and systems, as suggested in this paper, stresses the importance of co-evolving the systems, as soon as users evolve the performance of their tasks. Co-evolution of users and systems is rooted in the usability engineering, in that it supports designers to collect feedback on system from the field of use, to improve the system usability [10]. However, recent works claim that co-evolution requires specific analysis and evaluation activities [16].

In [5] we proposed an approach for automated support to user and system co-evolution. We started from the definition of the concept of *interaction pattern*. Informally speaking, interaction patterns are sequences of activities which the user of an IS performs in some specific situation during the interactive execution of a task. Therefore interaction patterns can be regarded as a representation of cognitive structures bringing knowledge about the interactions of a user with a system. IS designers are interested in recognizing these patterns and the reasons of their repetition, so that they can evaluate if it is worthwhile to co-evolve the system, for example by the introduction of new functionalities. An interaction pattern can be observed and expressed in a form suitable for subsequent automatic analysis. Assuming the knowledge of the control automaton of the IS, during the interaction it is easy to observe the activity $a(t)$ performed by the user at time step t and relate it to the current state $s(t)$ of the control automaton, and then derive the state $s(t+1)$ which is reached as a consequence of $a(t)$.

A prototypal system, called SIC (Supporting Interaction Co-evolution), has been implemented to support observation and recognition of interaction patterns. An extended description of SIC is provided in [5]. We give here a brief sketch of its architecture. It consists of an Interaction Observer, two Recognition Agents and some designer support tools. The former component is in charge of observing user activities, with the purpose of storing observed sequences in a log file, which is written following the XML standard for document description, to facilitate interoperability and document exchange. The Recognition Agents analyze the log file and implement interaction pattern recognition techniques. In the developed prototype, two recognition techniques have been implemented: the first one devoted to the recognition of the system states preferred by the user and the second devoted to the recognition of recurrent user behaviors. The agents then exchange messages with the designer in order to notify interaction patterns. Finally, the designer may modify the IS using the co-evolution support tools, which facilitate this activity.

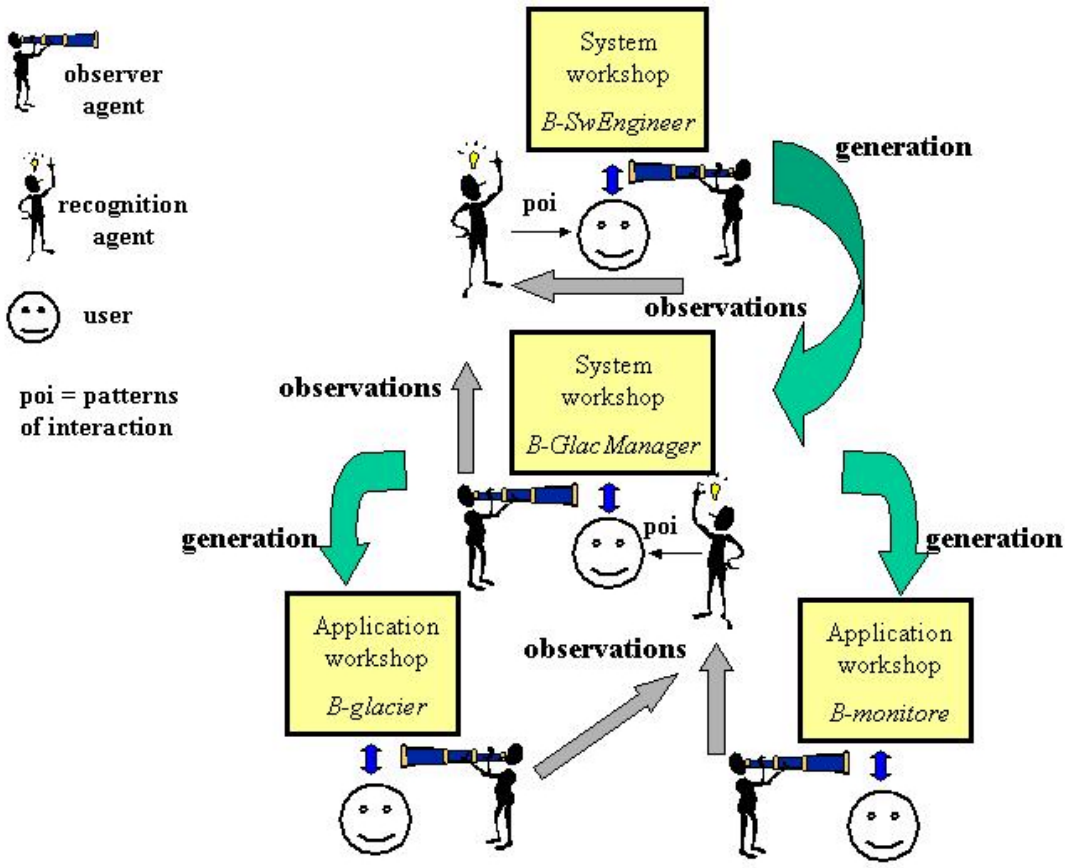

Figure 2. An agent-based architecture supporting co-evolution.

The organization of SIC naturally lends itself to a multi-agent system. One of the main advantages of this type of organization is its openness to extensions, by the addition of new specialized agents. For example the set of recognition agents might be extended in order to include further recognition techniques. For this reason SIC has been implemented using MadKit [17], a Java-based platform which supports the development of multi-agent systems.

V. AN AGENT ARCHITECTURE SUPPORTING KNOWLEDGE MANAGEMENT IN INTERACTIVE SYSTEM DESIGN

The architecture we propose integrates the observation and extraction capabilities of the SIC prototype within the visual workshop hierarchy and is illustrated in figure 2. Each workshop in the hierarchy is associated with an agent (an *observer agent*) in charge of observing user activities and producing a log file to be transmitted to one or more *recognition agents* at the upper level.

The recognition agent(s) analyzes the log files and extracts interaction patterns to be submitted to the attention of the professional in charge of managing that level in the hierarchy. While s/he uses the relevant workshop to introduce the changes possibly required by the interaction patterns emerged, his/her work is in turn the subject of observation by the relevant observer agent so that the same process can be repeated with respect to the upper level.

Let us observe that the observations collected at the highest level can only be managed at the same level. Therefore, the software engineer is also in charge of evaluating and evolving his/her own interaction with *B-SwEngineer*.

The adoption of a multi-agent architecture is appropriate in this context for several reasons:

- the workshops corresponding to different levels of the hierarchy are, in general, physically distributed so that the communication and cooperation mechanisms typically provided by a multi-agent architecture are required to share observation and analysis results;
- observation and recognition activities require some degree of autonomy by the software entities in charge of carrying out them, since they can not, in general, be carried out in a supervised manner;
- the openness of a multi-agent architecture eases the adoption of adaptive incremental techniques, based on the introduction of further classes of agents, for instance, it can be imagined that the results produced by a recognition agent can be communicated to a modification agent, able to automatically formulate IS modification proposals and to submit them to the designer.

A prototypal implementation of the proposed architecture is currently under development using BANCO [13].

BANCO is an XML specification of a visual environment, which can be accessed and interpreted by every public available SVG compliant browser. SVG is W3C recommendation for vector graphics. The browser accesses BANCO receiving from the server messages, written in a XML dialect called BML (Banco Markup Language), which also convey handles – as BML attributes - specifying how they can be shown and manipulated. BANCO embeds a kernel of scripting programs (about 50 KB) and a library of descriptive-customization rules [13]. The browser then interprets the scripting programs according to the descriptive-customization rules and displays the elements of BANCO interface. BANCO allows the implementation of the visual workshop hierarchy methodology because it is an XML document by which users can generate other XML documents. An expert can use BANCO to generate a new BANCO instance: this process is made possible by the structure of BANCO, which is modular, clearly separating the definition of interaction elements (and their organization and relationships), from their interpretation rules and interaction behaviors. Experts use BANCO at each level of the hierarchy to consistently generate workshops usable by the designate community of end-users. These workshops are instances of BANCO, each one specialized to its intended users and tasks.

As explained above, each instance includes an observer agent in charge of collecting information about user behavior. Non-leaf instances include also one or more recognition agents, which receive information from observer agents of the lower hierarchical level, apply analysis techniques and extract patterns of interaction, to be used by experts and/or designers for co-evolution.

VI. DISCUSSION

As stated in [8], questions such as "how often do users do X" or "how often does Y happen" are important for designers of an interactive system wishing to assess the impact of suspected problems or to focus development efforts for the next version. However this is not information that can be reliably collected in the usability lab. For this reason techniques for collecting information about system usage from the field and reporting it to the design and development team are recently emerging.

Ergolight [18] and the Quality Feedback System included in Netscape Communicator [19] are examples of commercial applications following this direction.

Ergolight analyzes logs of Web site navigation and generates exploratory diagnostic reports about the site usability. Since it operates on conventional log files generated by Web servers Ergolight carries out only global statistical evaluations mainly based on timing behavior of the user populations. For this reason it provides shallow indications of the kind "page difficult to find" or "page difficult to read" without giving design related information.

The Quality Feedback System included in Netscape Communicator to support beta-testing is based on a technology called TalkBack: it is a small piece of software in charge of gathering data about what is happening in Communicator whenever it crashes and automatically sending them to the Netscape development team. This technology focuses exclusively on software bug identification and characterization rather than on usability problems, but shares with our approach the idea of a sort of observer in charge of collecting relevant information on the user side and notifying them to the designers.

As to research proposals, a huge variety of techniques for extracting information either by direct observation or software logging of user behavior have been explored: an extensive survey is provided by [8]. Most of these proposals are conceived for supporting the execution of usability tests, to be carried out in predetermined circumstances (e.g. before the release of a system) rather than for supporting co-evolution along the whole system life-cycle. Moreover, as to our knowledge, none of these proposals encompasses a hierarchical organization of the professional figures involved in system development, while they simply consider the distinction between software designers and end-users. However, as stated in [20], software development involves many stakeholders representing many points of view. In particular, the definition of "end users" can even be confusing, since it may refer to people in charge of interacting with a software application but also to their colleagues, managers, and customers, who are otherwise affected by the deployment of an interactive systems.

Moreover, software developers are also end users namely of software tools.

Our architecture is in line with the requirements expressed in [20]: it provides a framework where existing observation and information extraction techniques can be applied in a more articulated context. In fact, the visual workshop hierarchy approach allows focusing knowledge extraction and manipulation on specific user communities, reflecting their different roles. Workshops within the hierarchy may be evolved in a separate though in a coordinated fashion.

The architecture we propose is based on agent technology.

In fact, according to the definition provided in [21], agents are characterized by the following properties:

- *autonomy*: an agent must have some kind of control on its actions and its internal state in order to be capable of carrying out its work without human intervention;
- *social ability*, it must be able to interact and cooperate with other agents by using some kind of agent communication language;
- *reactivity*: it perceives its environment and must cope appropriately and in a timely fashion with changes occurring within it;
- *pro-activeness*: it should be able to exhibit goal-directed behavior, i.e. it should not simply act in

response of its environment but it should "take the initiative" in relation to its internal needs and current mental state.

In our approach, the software entities devoted to system observation and evolution need all these properties:

- they must be able to autonomously observe interactions and find patterns of user behavior without being under the direct control of a human;
- they must be able to communicate their results to the agents at the higher levels in the hierarchy or to a human agent (i.e. the designer);
- they must be reactive in the sense that they must perceive user activities and carry appropriate actions on their basis;
- finally, they must be pro-active with respect to the designer by notifying him/her whenever interesting (e.g. anomalous) interactive patterns have been observed.

Use of agents able to perform observations of user interaction is advocated in other proposals, e.g. APE [22] and EDEM (Expectation-Driven Event Monitoring) [23].

The architecture of APE (Adaptive Programming Environment) is constituted by three software agents, an Observer, an Apprentice and an Assistant. The Observer monitors user's actions and stores them into a trace. The Apprentice applies machine learning techniques to learn situation patterns in which repetitive tasks are performed, with the purpose of building a set of user's habits. The Assistant proposes to the users the performance of repetitive tasks whenever user's actions match one or several learned situation patterns. APE agent architecture and its operation have some similarity with our proposal but do not consider a hierarchical organization of interactive environments.

EDEM is an agent-based system, which collects usage and contextual data and exploits a multi-level event model in order to compare developer's usage expectations against actual usage, at different levels of abstraction. Expectation mismatches are reported to the designer. A possible limitation is the fact that designers may not be able to correctly figure out expectations. The fact that the designer has some expectations on user behavior and that these expectations might be disappointed is based on the hypothesis that the designer already possesses adequate knowledge about the usages of the system Our approach focuses rather on the acquisition of knowledge from actual usages, possibly not foreseen by the designers.

VII. Conclusion

We have presented an agent-based architecture to support knowledge management in the life cycle of interactive systems. The proposal complements the approach of visual workshop hierarchy recently proposed in [13] with a hierarchy of agents providing the techniques supporting co-evolution analyzed in [5].

The feasibility of the approach is demonstrated by the initial prototyping activity carried out using a recently introduced technology, called BANCO, which turns out to be particularly suited for implementation of our ideas.

The main goal of future work is the development of a complete hierarchy in a working environment: we are currently analyzing a case study in the industrial automation field.

References

[1] D. Schuler, A. Namioka, Preface, *Participatory Design, Principles and Practice*, Schuler D., Namioka A. eds., Lawrence Erlbaum Ass. Inc.Hillsday, vii, N.J, 1993.

[2] J. Murray, D. Schell, C. Willis, User Centered Design in Action, *Proc. SIGDOC 97*, Acm, N.Y., 1997, 181-188.

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

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

[5] S. Arondi, P. Baroni, D. Fogli, P. Mussio, Supporting co-evolution of users and systems by the recognition of Interaction Patterns. *Proceedings of the International Conference on Advanced Visual Interfaces (AVI 2002)*, Trento (I), May 2002, 177-189.

[6] A. Dix, J. Finlay, G. Abowd, R. Beale, *Human Computer Interaction*, Prentice Hall, London, 1998.

[7] M. Petre, T.R.G. Green, Learning to Read Graphics: Some Evidence that 'Seeing' an Information Display is an Acquired Skill. *Journal of Visual Languages and Computing*, 4(1), 1993, 55-70.

[8] D. M. Hilbert, D. F. Redmiles, Extracting usability information from user interface events. *ACM Computing Surveys*, 32(4), 2000, 384-421.

[9] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, T. Carey, *Human-Computer Interaction*, Addison-Wesley, UK, 1994.

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

[11] J. Nielsen, R. L. Mack, K .H. Bergendorff, N. L. Grishkowsky, Integrated software in the professional work environment: evidence from questionnaires and interviews, *Proc. CHI 86 Conference*, Boston, MA, 1986, 11-120.

[12] P. Carrara , A. Rampini, An event-based archive of soft maps for the analysis of glacier changes, in *Proc. of SIS, 15th Int. Symp. Informatics for Environmental Protection*, Zurich (CH), October 2001, 395-402.

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

[14] J. M. Carroll, M.B. Rosson, Deliberated Evolution: Stalking the View Matcher in design space. *Human-Computer Interaction* 6 (3,4), 1992, 281-318.

[15] J. Brown, T.C.N. Graham, T. Wright, The Vista environment for the coevolutionary design of user interfaces. *Proc. of CHI 98, Conf. on Human Factors in Computer Systems*, Los Angeles, 1998, 376-383.

[16] G. Bourguin, A. Derycke, J.C. Tarby, Beyond the Interface: Co-evolution inside Interactive Systems - A Proposal Founded on Activity Theory, *Proc. IHM-HCI 2001*.

[17] MadKit Web Site, http://www.madkit.org.

[18] Ergolight Usability Software, http://www.ergolight-sw.com.

[19] Netscape Quality Feedback System for Netscape Communicator 4.5, http://wp.netscape.com/communicator/navigator/v4.5/qfs1.html.

[20] D. F. Redmiles, Supporting the end users' views. *Proceedings of the International Conference on Advanced Visual Interfaces (AVI 2002)*, Trento (I), May 2002, 34-42.

[21] M. Wooldridge, N. R. Jennings, Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, vol. 10(2), 1995, 115-152.

[22] J-D. Ruvini, C. Dony, APE: Learning User's Habits to Automate Repetitive Tasks. *ACM Int. Conf. on Intelligent User Interfaces*, New Orleans, LA, USA, 2000, 229-232.

[23] D. M. Hilbert, J. E. Robbins, D. F. Redmiles, EDEM: Intelligent Agents for Collecting Usage Data and Increasing User Involvement in Development. *ACM Int. Conf. on Intelligent User Interfaces*, San Francisco, CA, 1998, 73-76.