Supporting co-evolution of users and systems by the recognition of Interaction Patterns

Stefano Arondi, Pietro Baroni, Daniela Fogli, Piero Mussio

Dipartimento di Elettronica per l'Automazione
Università di Brescia
Via Branze 38, 25123 Brescia Italy
Tel. + 39 030 3715450 Fax + 39 030 380014

{baroni, fogli, mussio}@ing.unibs.it

ABSTRACT

This paper presents an approach to support the designer of Visual Interactive Systems (VISs) in adapting a VIS to the evolution of its users. This process is called *co-evolution* of users and systems. The approach is based on the identification of the patterns of interaction between the user and an interactive system and on their use for the evolution of the system to facilitate novel usages introduced by the user. The approach is focused on WIMP systems and is based on the recently introduced PCL (Pictorial Computing Laboratory) model of interaction, within which we provide a novel definition of *interaction pattern*. The proposal assumes that the VIS is observed by an external system called SIC (Supporting Interaction Co-evolution), which is in charge of recording the interactions between the user and the VIS and of analyzing the relevant interaction patterns. In particular, SIC exploits a UML-based statechart specification of the VIS in order to associate observed user activities with the states of the interactive process. This information provides a useful basis for a variety of pattern recognition techniques. Two techniques called *usual state* and *recurrent sequence recognition* are illustrated and the results of a first experiment are discussed.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques – *user interfaces, state diagrams.*

General Terms

Design, Experimentation, Human Factors, Languages.

Keywords

Visual interface design, visual sentence, co-evolution, system observation.

1. INTRODUCTION

An intriguing phenomenon, often observed in (Human-Computer Interaction) HCI studies, is that "using the system changes the users, and as they change they will use the system in new ways" [24]. 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. This paper presents an approach to support co-evolution, helping designers in identifying the new patterns of user behavior and the context in which they occur, so that designers can decide if it is worthwhile to co-evolve the system. The concept of user behavior in a context will be made precise by the definition of *interaction pattern*. This notion will be framed in a model of interaction and a formalism for its description, recently introduced by the Pictorial Computing Laboratory (PCL) [6]. Interaction patterns are identified by a system, SIC (Supporting Interaction Co-evolution), able to observe the interactions between the user and an interactive system. SIC associates user activities with the states of the interactive process, exploiting a statechart [15] specification of the system, in the notation adopted in UML (Unified Modeling Language) [4], and uses XML (Extensible Markup Language) [31] to document its activity and manage the interaction with the designer. The identified interaction patterns are thereafter examined by the designer. Two examples of co-evolution, based on *usual state* and *recurrent sequence patterns* identified by SIC, are discussed with reference to a running example relevant to an experiment on an interactive prototype, called "Online Bookshop".

2. CO-EVOLUTION AS THE RESULT OF USER-DESIGNER INTERACTION

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 version of spreadsheets, after the observation that users frequently forced the spreadsheet to manage non-calculation data for data archiving and other tasks [25]. Several examples derive from the observation of users learning how to interact with web documents. At present, we are facing an astonishing spread and a continuous discovery of new applications, which call for a technological evolution, including also semantic hints of the content [3].

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 this paper, co-evolution is modeled as emerging from the interaction between users and system designers, mediated by software tools (figure 1). A user performs some task interacting with an application Ap - in the running example the demonstrative prototype “Online Bookshop”. The interaction occurs through a video screen on which sequences of images are shown at successive instants of time. Interacting with Ap, users may find how to use Ap in achieving new tasks or acquire some habits.

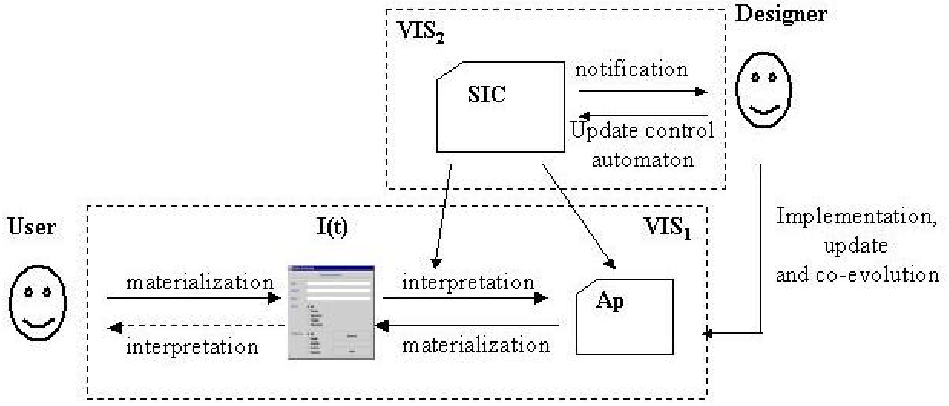


Figure 1: Co-evolution as an interaction between user and designer mediated by two VISs.

These findings of the users result into *patterns of interaction*, sequences of activities which users execute when some specific situation arises during the task at hand. The interaction is monitored by SIC, a system which detects both the user activities and the situations in which they occur. SIC records this information in a log file, extracts interaction patterns from it and notifies them to the designer. The designer decides if it is worthwhile to evolve the application Ap. To make these concepts more precise, in the next section we revise the PCL model and formalisms for specifying Visual Interactive Systems, and adapt them to the co-evolution needs.

3. A MODEL OF THE HUMAN-COMPUTER INTERACTION PROCESS

In the PCL approach [6], HCI is modeled as a process in which the user and the computer communicate by materializing and interpreting a sequence of messages at successive instants of time. Two interpretations of each image on the screen arise in the interaction: one performed by the user achieving the task, depending on his/her role in the task, as well as on his/her culture, experience, and skills, and the second internal to the system, associating the image with a computational meaning, as determined by the programs implemented in the system [6].

If we restrict to the case of WIMP (Windows, Icons, Menus, Pointers) interaction [12], the messages exchanged are the whole images which appear on the screen display of a computer and are formed by text, icon, graphs, pictures, windows. Figure 2a shows one of these images, representing the initial message of the demonstrative prototype “Online Bookshop”, designed to support a human in choosing and purchasing books.

The HCI process is viewed as a sequence of cycles: the human detects the image on the screen, derives the message meaning, decides what to do next, and manifests his/her intention by an activity performed operating on the input devices of the system; the system perceives these operations as a stream of input events, interprets them, computes the response to human activity and materializes the results on the screen, so that they can be perceived and interpreted by the human. In principle, this cycle is repeated until the human decides that the process has to be finished, because the task has been achieved or failed.

In each cycle the system plays several roles. First it is the medium conveying the messages on which interaction is based: some of its programs and tools generate, maintain and display the images on the screen. Second, the system plays the role of the second interacting entity, denoted by Ap: some of its programs compute the system reaction to the user activity. The input to Ap are computed by a set of devices and programs which capture and digitize the input operations performed by the human, relate them to the image on the screen and assign them a meaning: this third set of devices and programs plays the role of the tools used by the human to manifest his/her intention. Last, there are some programs which materialize the results of the computation performed by Ap, determining the image on the screen, acting as the materialization tool for Ap. A Visual Interactive System (VIS) is a software system which organizes the four sets of programs, acting as image support, second communicant, and I/O tools. In figure 1 two VISs are schematized: the first one allows the interaction of the user with Ap, which acts as second communicant, while the second VIS (VIS_2) allows the interaction of the designer with SIC, which acts as second communicant. The PCL approach formally describes this model.

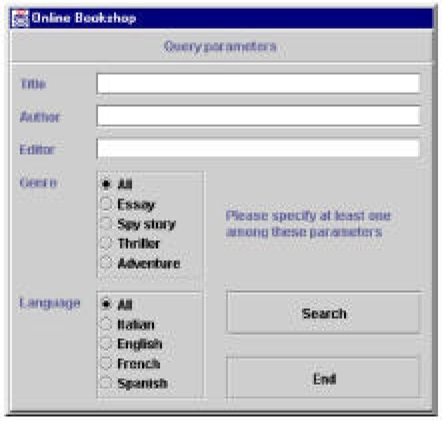


Figure 2a: The initial message of “Online Bookshop”.

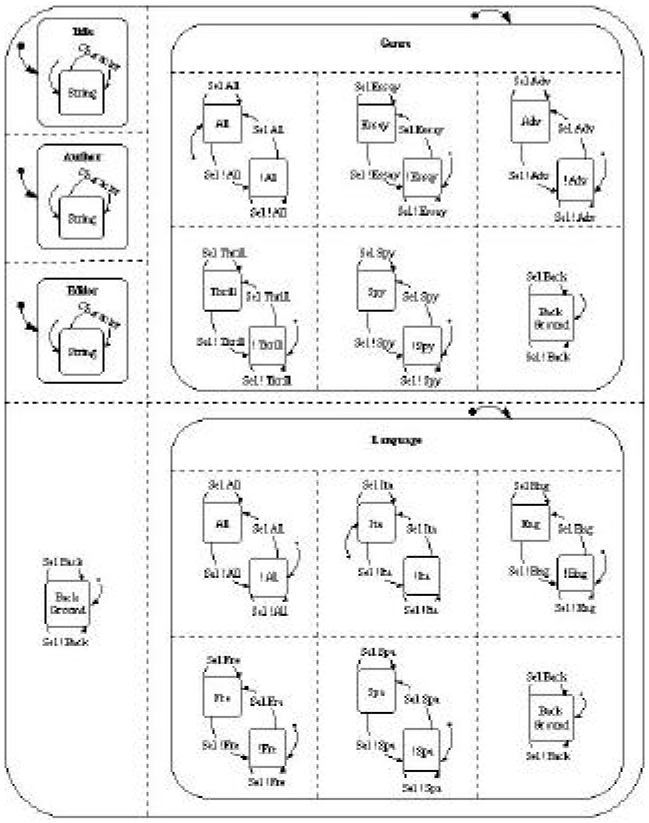


Figure 2b: UML statechart sketching the core description of the initial message of “Online Bookshop” (“Search” and “End” buttons are omitted).

4. THE STATE OF THE INTERACTION PROCESS

Humans interpret an image on the screen by recognizing *characteristic structures* (css or structures for short), i.e. sets of image *pixels* which they perceive as functional or perceptual units. The cs recognition and interpretation results into the association of a meaning with a structure. Identification and interpretation (or misinterpretation) of css are influenced by the similarity (dissimilarities) with real tools and graphical constructs traditionally adopted by the user community to perform and document their activity. For example, in figure 2a a user who is acquainted with domestic appliances and fill-in forms recognizes two sets of (virtual) radio buttons, two buttons and three fields to be written in. The human deduces the meaning associated with the whole image by combining the meanings of the css recognized in it.

On the system side, each cs on the screen is the result of a computational process, which determines its graphical features, position, persistence and behavior and is also in charge of managing the inputs in the interaction process which are related to it. A cs is the pictorial part of a virtual entity, determined by the computational process. For example, the rectangle framing the word "Search" in figure 2a is a cs representing a virtual button, which can be selected by the human positioning the mouse pointer on it and then clicking a mouse button. The selected button changes its graphical properties to manifest the new state of the generating process, and fires an associated computational process. In a well designed system, at each step of the interaction each cs hints the state of its underlying process, conveying information for the user to understand the state of the process with reference to the task being executed. The current state of the underlying process is described by the triple <cs, u, <int,mat>>, where a) cs is the set of pixels produced by the process on the screen; b) u specifies the current state of the computational process generating the virtual entity (e.g. the program being executed, the state of its memory, and the pointer to the current instruction); and c) the pair <int, mat> are two functions describing the correspondences between elements of the cs and components of u. In the PCL model [5], this triple is called "characteristic pattern" and denoted as cp=<cs,u,<int,mat>>. The cp is a specification of the entities called interactive objects or widgets [12].

The state of the whole machine participating to the interaction process is specified in a similar way, reflecting the necessity of the machine to play its different roles: the specification has therefore to take into account the current state of the whole message – i.e. the current image on the screen -, the state of Ap as well as the relations between what appears on the screen and the processes going on within the system. The state of the overall process is described as a triple vs=<**i**, **u**, <**int**, **mat**>>, where **i** is the array of pixels constituting the current image, **u** is a suitable description of the current state of the process Ap, **int** and **mat** are two functions relating elements of **i** with components of **u**, formalizing the interpretation and materialization arrows of fig. 1. This triple is called "visual sentence". Figure 2a displays an image **i** of a vs and figure 2b its associated **u** sketched in UML notation [4], as a set of orthogonal sub-states, collectively describing the current state of Ap. The int and mat functions are not represented due to lack of space. According to this definition, a visual sentence is a special cp whose cs is the whole image on the screen, i.e. its image part **i** is the current message to be interpreted by the human and by the system. Figure 3 shows the complete definition of the cp of a radio button displayed using the same techniques.

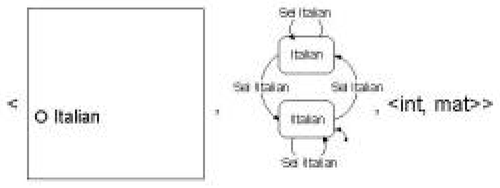
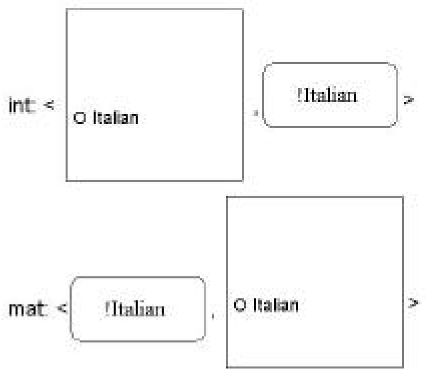


Figure 3: The cp of the radio button "Italian".

The int and mat functions of the cp in figure 3 are as follows:



Looking at figure 2, it can be noted that the cs and u of the radio button of figure 3 appear as elements of the image **i** and of the description **u** of the visual sentence. Actually, as discussed in [13], cps can be composed to form more complex ones. In the design of a VIS, a finite set of equivalence classes of cps, called atomic classes, and a set of rules for their instantiation and combination are defined from which more complex cps, up to vss, are derived. In the following, the set of atomic classes of cps is denoted as CP. A programmed implementation of the equivalence classes and of the rules can be naturally produced through a toolkit [12].

5. THE DYNAMICS OF THE INTERACTION PROCESS

In each interaction cycle, a visual sentence vs_1 is transformed into a visual sentence vs_2 as the consequence of some human activity a. In a WIMP system, the human performs an activity operating on an input device – say clicking a mouse button – in relation to some cs recognized on the screen. The system relates the operation to the current active cs (in WIMP systems, the one pointed to by the mouse pointer) – e.g. the virtual button "Search" in figure 2a – and interprets it as a command from the user – push the search button. Then it changes the button state and fires the consequent computation, referred to in the u associated to the cs in the corresponding cp. The computation actually consists in searching the books whose characteristics are defined in the text fields and radio buttons.

The designer describes the human activity as a pair a=<operation, cp> and specifies the transformation as tr:<a,<vs_1,vs_2>>, where cp in a is present in vs_1. The interaction process is specified as a sequence of such transformations. An interaction process cannot be predicted in advance, because it is determined by the sequence of activities the human decides to perform and each decision is taken on the basis of the observation of the current message shown by the system. However a phase space for a VIS can be defined, if an initial state vs_0 is specified. This space is constituted by all the (in general) infinite sequences of transformations which can be generated by humans interacting with the VIS, starting from vs_0.

An approach to derive a finite specification of a control automaton governing the interaction is discussed in detail in [6] [13]. Here we present a short account of the methods.

The method exploits the finitary nature of the definition of cp. A cp is the description of the state of a computational element u in a precise step of the computation. This description is finite, i.e. it requires a finite amount of memory to be stored. However, it can be enlarged by a finite amount in the next step of the computation. The vs, being a special cp, is also finitary. The memory required to describe the interaction process from this abstract point of view is unbound, finite in each step of the computation, but amenable to a finite increment in the next step.

Moreover, in any step of the interaction, the user can perform a finite number of operations on the current vs, those allowed by the input devices of the system, which are finite in number and each one able to generate a finite number of digital events, perceivable by the machine. The set of css appearing on the screen is also finite: as a consequence the set of activities the human can perform on a vs are finite. Moreover two activities a_1 and a_2 are equivalent if $operation_1$ and $operation_2$ correspond to the same set of digital events and cp_1 and cp_2 belong to the same class. The number of equivalence classes of activities is finite due to the finite number of equivalence classes in CP and of digital events. In the following, each activity equivalence class is denoted by a representative activity a belonging to it. The set of activity equivalence classes is denoted by A. Two vss are equivalent ($vs_1 \equiv vs_2$), if the human can perform on them the same set of classes of activity. Due to the finiteness of the number of classes of activity, also the cardinality of its powerset, i.e. the maximum number of the classes of equivalence on the set of vss, is finite. The set of vs equivalence classes is denoted by S.

Let us restrict to deterministic VISs, i.e. systems in which each activity a performed on a visual sentence vs_1 determines one and only one transformation tr:<a,<vs_1,vs_2>>. Let us also assume that, when switched on, the VIS is in state vs_0. A function $f(a,vs_1) = vs_2$ can be defined, mapping each pair <activity a, visual sentence vs_1> into a visual sentence vs_2, where vs_2 is the result of the transformation tr. The binary relation $\equiv$ on S is a congruence relation with respect to f.

A finite state machine – called Control Automaton (CA) - can be defined as follows on the input alphabet A. The set of states is the set of vs equivalence classes S. The start state s_0 is the class to which vs_0 belongs to. The transition function is of the form $g(a,s_1)=s_2$, where s_1, s_2 belong to S and a is in A. CA is equipped with an output function in the form $U(a,s_1)=nu$, where nu indicates the computational element of VIS which computes the transformation. Figure 6 shows an example of CA, referred to the "Online Bookshop" prototype. The CA is specified using the UML statechart notation, activities are described at the *window system level*, i.e. according to [16] input device events are associated to windows and widgets on the screen, and computational constructs are hinted by their names [7]. Each state of the automaton is a macrostate in the statechart language, i.e. it can be expanded into a more refined representation: for instance, the macrostate "InsQueryMask" can be expanded in the UML statechart of figure 2b. From the point of view of the interaction, each macrostate is associated to a set of vss, which are equivalent with respect to the user activities, whose name are labels for the transitions going out of the macrostate. These vss share a common part of the image, which is here called 'mask'. The names of the macrostates refer to these common structures, which indicate to the users the current state of the machine and recall them the sub-task that they are performing.

6. PATTERNS OF INTERACTION

Patterns of interaction (poi) are sequences of activities which the user performs in some specific situation during the interactive execution of a task. 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. A pattern of interaction however cannot be predicted in advance, because it is determined by the users findings in the execution of some task; but it can be observed and it can be expressed in a form suitable for subsequent automatic analysis. During the interaction it is easy to observe the image i on the screen as well as the activity a(t) performed by the user at time step t, if one knows the set CP. Restricting to deterministic VISs and assuming the knowledge of CA, it is possible to derive the state s(t) of the CA to which the current vs(t) belongs to and, observing the activity a(t) performed by the user, derive the state s(t+1) to which vs(t+1) belongs to and which is reached as a consequence of a(t).

It is therefore possible to track, log and analyze the path followed by the CA as a result of the sequence of activities performed by the user. The state s(t) records in a synthetic way the history of the interaction, i.e. resumes the situation. In summary, if the following conditions are satisfied: a) the system is deterministic, i.e. only one u corresponds to each image i, b) the CA specifying the VIS is known, c) the set CP of the classes of atomic cps is known and d) the set A of the classes of user activities is known, then the interaction process can be described as the observed sequence of pairs <s(t), a(t)>, where s(t) describes the state of the control automaton before the activity is carried out and a(t) describes the performed activity. A *pattern of interaction* is a sub-sequence of an interaction process and is described as a finite sequence of pairs <s(t), a(t)>, with t in [t_1,..,t_n].

7. CO-EVOLUTION OF VISUAL INTERACTIVE SYSTEMS

The co-evolution of a VIS consists in three steps: *observation of the interaction*, *recognition of interaction patterns* and *adaptation of the VIS*. The first two steps are carried out by SIC, which provides the necessary information to the designer, who performs the third step. We assume that SIC has access to a representation of the CA specifying the VIS, to the set CP of atomic classes of cps and to the set A of classes of user activities.

7.1 An architecture for co-evolution

A prototypal system, called SIC, has been implemented to support observation and recognition of interaction patterns as described in the previous section. The high level architecture of the current implementation of SIC is shown in the left part of figure 4. 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 [31], to facilitate interoperability and document exchange. The Recognition Agents analyze the log file and implement the two interaction pattern recognition techniques described in the next sub-sections. They then exchange messages with the designer in order to notify interaction patterns. Finally, the designer may modify the VIS (e.g. adding a new transition in the CA) using the co-evolution support tools, which facilitate this activity.

The architecture of SIC is particularly suitable for an agent-based implementation [19]. Generally speaking, an agent is a computational entity which is situated in an environment where it autonomously pursues its goals and is able to exhibit both reactive and proactive behavior. 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 [21], a Java-based platform which supports the development of multi-agent systems.

7.2 Observation of the interaction

System observation is in charge of appending a new pair <s(t), a(t)> to a log file each time a user activity is performed. Moreover, the observer computes the new state s(t+1) of the automaton to be used in the processing of the next observed activity.

The algorithm describing the observation activity is as follows:

```
Interaction Observation
CurrentState := s0;
Do
   Let CurrentActivity := Next observation of
   user activity;
   Append couple <CurrentState,
   CurrentActivity> to logfile;
   Update CurrentState according to
   CurrentState and CurrentActivity on the
   basis of the control automaton;
While CurrentState ≠ TerminationState
```

7.3 Recognition of interaction patterns

The recognition of interaction patterns is carried out taking into account the control automaton of the VIS in order to properly interpret the contents of the log file. The results of the interaction pattern recognition can be used by the designer to identify useful modifications of the control automaton, which give rise to an evolved version of the interactive system. In particular, to experiment the potential of our approach we have considered two kinds of interaction pattern recognition, that we have called *usual state* and *recurrent sequence recognition*.

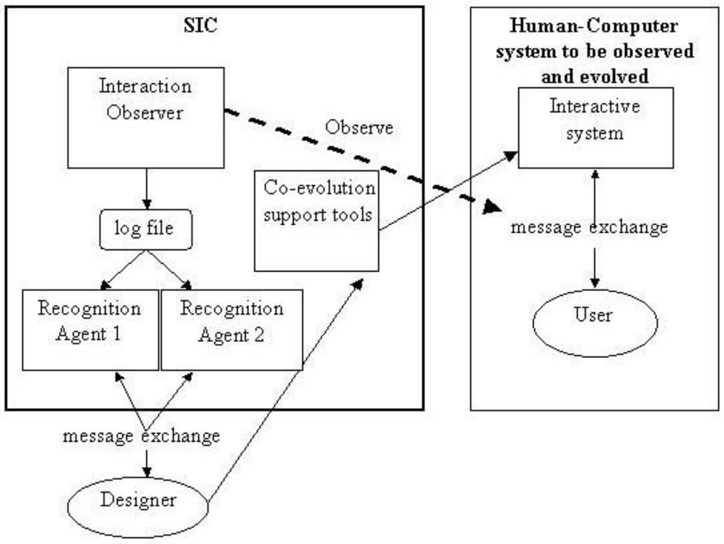


Figure 4: Current SIC Architecture and its interactions with the human-computer system.

7.3.1 Usual state recognition

The usual state recognition technique is used to identify those values of variables defining the state of the VIS, which are more suitable to be assumed as default values. The aim of the technique is to allow the designer to establish default values which make more efficient the achievement of the task by the user. To this end, we classify cps in CP (atomic cps) in two categories: a) *data insertion cps*, devoted to establishing the values of the variables which determine the VIS behavior; b) *action cps*, devoted to firing computational activities of Ap. In figure 5, examples of cps of the first category are check boxes, radio buttons and text boxes, while examples of the second category are buttons.

An important sub-category of action cps are the *exit cps*, which determine the transition from a macrostate to another macrostate of CA (figure 6), called here *exit transition*. For example, in figure 5, "Order" and "Cancel" buttons are exit cps.

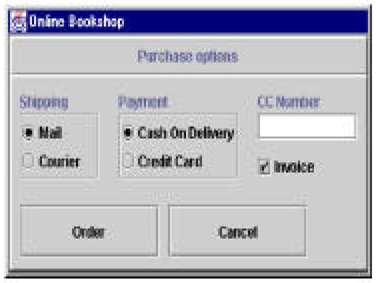


Figure 5: The mask of "Online Bookshop" for payment and shipping.

An exit transition can be associated with three types of activity: a) *confirmation activity* (for example, in figure 5, a mouse click on the "Order" button), which starts the subsequent processing activity; b) *cancellation activity* (for example, in figure 5 a mouse click on "Cancel" button), to return to the previous mask; c) *quit activity* (for example, a click on the "End" button in figure 2a), to definitely exit the program. The user triggers a confirmation activity when s/he is satisfied with the set values of the variables on which Ap has to perform its computations. This set of values are the states of the data insertion cps. We define *usual state* the state of a data insertion cp which is most frequently selected by the user before triggering a confirmation activity.

The usual state recognition technique is in charge of identifying the usual states in order to notify them to the designer, if different from default values. The designer may then decide to set them as default values by modifying the initial state markers (arrows with bold circle queue in figure 2b) which are present in the statechart representation of the control automaton.

In practice, the usual state recognition technique analyzes the log file by looking for those pairs <s(t), a(t)>, where the user activity a(t) corresponds to a confirmation activity. A frequency analysis can be performed on the states of the cps in s(t), so to determine usual states. Information about usual states is presented to the designer, who then has to decide whether modifying default states accordingly. The algorithm for usual state recognition is as follows:

```
Usual state recognition
For each macrostate
   For each data insertion cp
        For each state assumed by the cp
             Create a counter associated to it
For each pair <s(t), a(t)> in logfile
   If a(t) is a confirmation activity
        Then For each sub-state in s(t)
              referring to a data insertion cp
                increment the counter associated
                to the sub-state
For each macrostate
   For each data insertion cp
        Find the maximum value among the
        counters associated to the states of
        the cp
        If the maximum value is not associated
        to the default state
                Then notify it to the designer
```

7.3.2 *Recurrent sequence recognition*

The second technique we considered is related with the recognition of recurrent sequences within the log file. Several techniques for sequence retrieval have been proposed [17]. In the context of the interaction model adopted in the present work, the result of observation activity (actually, the log file) is a string on the alphabet ALP whose elements are pairs <state, user activity>, i.e. ALP = S x A. Projections of this string can be obtained by restricting the alphabet to S or A only, i.e. focusing attention on states or activities respectively.

A retrieval technique can be applied to find recurrent sub-strings within the complete log file or a projection of it. The algorithm for sub-string retrieval is parametric with respect to maximum sub-string length (MSL), minimum number of sub-string occurrences considered significant (MNS), and the selected alphabet (either S x A, S, or A). Frequent sub-strings are identified by constructing a tree data structure similar to those used in data compression algorithms (see [20] for details).

The algorithm works as follows:

```
Recurrent sequence recognition
If the selected alphabet ≠ ALP
   Then let P = projection of the log file
         wrt the selected alphabet
   Else let P = contents of log file
Initialize an empty tree structure T
For L := 2 to MSL
   For each element E in P
      Extract from P the sub-string of
      length L starting with E
      If the sub-string is in T
              Then   update the corresponding
                     occurrence counter
              Else   add the sub-string to T
                     and set its occurrence
                     counter to 1
   End (internal for)
   Select from T all the sub-strings whose
   occurrence counter value is over MNS
End (external for)
Notify the selected sub-strings to the
designer
```

Let us now consider an example of system co-evolution related with recurrent sequence recognition. Many systems ask the user whether saving work before definitely exiting the application after the user has given an “Exit” command, if any modification has been introduced since the last “Save” command. If the user is accustomed to save his/her work after the “Exit” command, it can be recognized that the relevant sequence of commands is rather frequent. In this case the designer might decide to add a new command “Save and Exit” among the exit choices (by the way this command was available in the UNIX *vi* editor, while it is not present in MSOffice applications).

An example of the implications of the use of different alphabets can be noticed in this case. In fact the recurrent sub-string formed by activities “Exit” and “Save” can be identified if the considered alphabet is A only, possibly resulting in the addition of a new activity, available in all states. On the other hand, if the alphabet S x A is considered, the additional information about the states can be exploited to make the new command available only in the states where it is actually needed.

A significant support to the system designer can be given by the recognition of *recurrent anomalous sequences*. A very simple example of this case is a sequence of mouse clicks on the background, which might indicate that the user finds the interface unclear or hard to understand. The identification of such kind of sequences may in general help the designer in finding possible design mistakes or problems.

8. EXPERIMENTAL ACTIVITY

As already mentioned, experimental activity has been carried out on “Online Bookshop”, a prototypal VIS simulating the on-line purchase of books, which was designed according to a precise discipline so that its CA was available as its specification. In the prototype, vs_0 is the vs presented in figure 2a, where the user is prompted for his/her book preferences. The user activates the related search by clicking on the “Search” button. In this case, a sequence of vss showing search progression are presented. When the search is completed or interrupted by the user, the vs presenting the (possibly partial) search results is shown. The user can then select one or more books and require their purchase. In this case, a mask requiring payment and shipping information is shown (figure 5). After the user has given and confirmed the required information, a notification that the purchase has been completed is presented. The user can then proceed to further purchases or to a new search.

8.1 An example of usual state recognition

Various working sessions have been carried out in the experiments, so leading to the creation of a log file storing the information about the history of the interactions between the users and the VIS. The log file has been analyzed by one of the recognition agents, which has then notified to the designer the usual states of some of the cps in the VIS. For example, in the experiments, the users had often activated the book search with the language “Italian” and the genre “Adventure”, and selected “Courier” for shipping and “Credit Card” for payment (figures 2a and 5). The system has suggested these choices as default values. It has to be noted that, even in this simple example, taking into account the system state is useful: in fact, only the state values associated with confirmation activities are analyzed. Considering information about user activities only would make impossible to distinguish between significant user choices and erroneous or wavering selection actions.

8.2 An example of recurrent sequence recognition

Recurrent sequence recognition has been applied to the log file referring to a set of working sessions with the prototypal system. Several sequences corresponding to expected user behaviors were extracted and presented to the designer, while one sequence turned out to be anomalous and required an additional analysis. The anomalous sequence is described by the highlighted transitions in figure 6, specifying the control automaton of “Online Bookshop”. As a matter of fact, the sequence corresponds to a relatively subtle design problem: after any purchase completion, the activity of pressing the “OK” button determines a transition to the previous mask (“PurchaseMask”), i.e. the one requiring payment and shipping data (figure 5). From that mask, the user always goes back to the mask presenting the results of the previous book search and then frequently goes back to the first mask to perform another search. This inefficient behavior was not noticed by the designer.

Actually, this behavior follows from a commonly adopted rule concerning the navigation between the masks of an interactive system: always going back to the previous mask in order not to disorient the user. However, this rule was not appropriate in this case and the notification of the recurrent anomalous sequence has permitted a suitable evolution of the system.

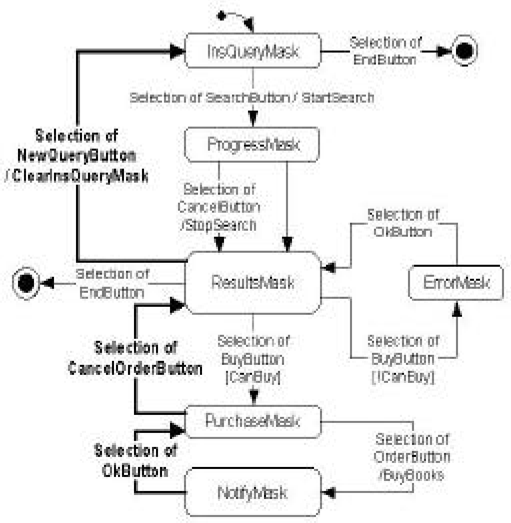

InsQueryMask
Selection of EndButton
Selection of SearchButton / StartSearch
Selection of NewQueryButton / ClearInsQueryMask
ProgressMask
Selection of CancelButton /StopSearch
Selection of OkButton
Selection of EndButton
ResultsMask
ErrorMask
Selection of CancelOrderButton
Selection of BuyButton [CanBuy]
Selection of BuyButton [!CanBuy]
PurchaseMask
Selection of OkButton
Selection of OrderButton /BuyBooks
NotifyMask

Figure 6: The UML statechart describing the CA of the first version of “Online Bookshop”.

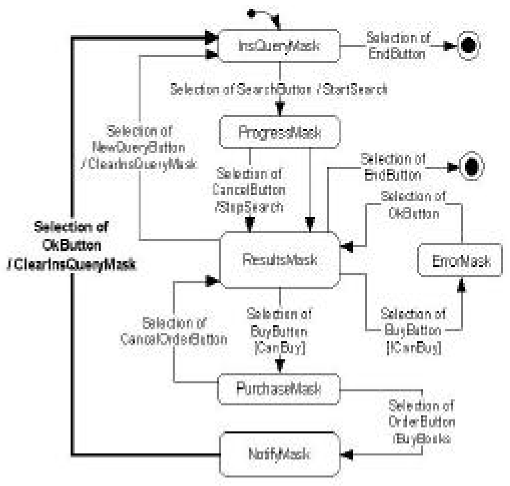

InsQueryMask
Selection of EndButton
Selection of SearchButton /StartSearch
ProgressMask
Selection of NewQueryButton /ClearInsQueryMask
Selection of CancelButton /StopSearch
Selection of EndButton
Selection of OkButton
Selection of OkButton /ClearInsQueryMask
ResultsMask
ErrorMask
Selection of CancelOrderButton
Selection of BuyButton [CanBuy]
Selection of BuyButton [!CanBuy]
PurchaseMask
Selection of OrderButton /BuyBooks
NotifyMask

Figure 7: The UML statechart describing the CA of "Online Bookshop" after the co-evolution.

Figure 7 shows the control automaton describing the system after this modification. It has to be noted that this kind of design problems is rather common, since general interface design rules admit exceptions, which may escape designer attention. The formal support provided by control automaton representation to interaction pattern recognition makes easier the identification of these problems.

9. DISCUSSION AND RELATED WORKS

Co-evolution is a word widely used in scientific works, and also in the HCI field, from Carroll and Rosson's co-evolution of users and tasks [9] 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 [8]. Co-evolution of users and systems as proposed 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 [24]. Tools designed to support co-evolution are suitable for observational evaluation in user-centered design approaches [27]. SIC observes the user behavior and context from inside the digital world; classical usability techniques [24] should be used to complement this view, feeding observation on system behavior in the context, from the human point of view. Use of agents, able to interview users and perform the two kinds of observation concurrently is foreseen in EDEM (Expectation-Driven Event Monitoring) [16]. 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. However designers may not be able to correctly figure out expectations. SIC focuses on the contrary on the observable user activity situated in the interaction context, described by the sequences of interaction patterns and analyses these sequences using the model of the interaction process, synthesized by the control automaton CA.

The model of HCI, from which the CA definition stems, derives its cyclic nature from the seminal model proposed by Hutchins et al. in [18]. However, the PCL model looks at the interactive system as a new medium [26] [1], and focuses on the communication as managed on the machine side, in order to derive an approach to the design of usable and co-evolvable interactive systems. From this point of view, the PCL model is indebted with the PIE (Program, Interpretation function, Effect) model [11], a black box model in which the difference between the effemeral display of the system and the permanent results is central to the definition of system properties. At the highest level of abstraction, the PCL model focuses on the messages exchanged without taking care of their specific nature, while at the lower level focuses on the system structure. It also recognizes the syndetic [2] nature of the system formed by the human and the computer – i.e. it recognizes the fundamental difference in the behavior of the two systems bound together in the interaction process. However, again the PCL model focuses on modeling the machine, taking into account the human as a source of constraints on the design and as a resource for its validation.

The specification of CA is a major difficulty in our approach. If the system has been properly specified, CA can be derived by the documentation or (in few cases) directly found in it. In other situations, the only possibility is to derive it from observations of the interaction with the system. Dealing with a similar problem, Guzdial characterizes sequences of user interface events by a Markov-based analysis and models the interaction process with nodes representing process steps and arcs indicating the observed probabilities of transitions between process steps [14]. This approach suggests the possibility of summarizing the partial knowledge obtained by the observations into a probabilistic CA, from which plausible deductions on interaction patterns can be obtained and used in co-evolution. However, Guzdial's approach, like most of those reported in the literature, focuses on the extraction and analysis of events generated by the user interface, without rooting observations into a model of the process itself (see [17] for a survey).

For instance, the system ActionStreams [23] aims at recognizing recurrent sequences of user activities and at inferring from them hierarchical models of user tasks. The approach does not take into account system states and, according to the author himself, "1) it does not match applications of a given task to different data, 2) it does not represent optional or conditional events, and 3) it does not coalesce sequences that differ trivially in the order of events".

As a different example, the APE (Adaptive Programming Environment) project [29] aims at developing a system able to help users in performing repetitive tasks. The architecture of APE 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 closely recall those of SIC, however situation patterns differ from our interaction patterns, as they do not include system state information.

Personalization issues have received a special attention in the context of Web systems, as witnessed by [28]. Many of these works focus on the personalization of contents to be presented to users (e.g. [22]). In particular, the approach presented in [10] adopts recent standards from the W3C Consortium, such as XML, RDF and P3P. In this proposal, a user agent at the client side captures the navigational history of the user and logs it as an XML file. Then, descriptive statistics are applied to the log file by executing a query (written in XML-QL) which produces the user profile in RDF. Finally, Web servers exploit user profiles, obtained from the clients according to the privacy rules supported by P3P, to deliver personalized information, namely information that fits user interests. While the application context and the goals of this approach are clearly different from ours, we share the design criterion of using standard representation formats (in particular XML and UML) for information exchanges among the components of the system. This is especially important in a multi-agent architecture where openness to future extensions and interoperability among agents, possibly developed by different people, are fundamental requirements.

The goals of the work presented in [30] are more similar to ours: a technique for discovery of navigation patterns, describing the routes followed by users while accessing web pages, is used to support a web designer in improving the pages of a site and the navigation links among them. Our work seems more articulated in the classification of interaction patterns and their structure, even if the study of the specific characteristics (and difficulties) of web systems is reserved to future research.

10. CONCLUSIONS

In this paper an approach is introduced to support user-system co-evolution based on the observation of the interactive process and on the recognition of interaction patterns. The approach is based on the PCL model of HCI: this makes possible a deeper analysis of the interaction process and more sophisticated forms of co-evolution with respect to approaches which are limited to a more shallow observation of user activity. The availability of a specification of the system control automaton is a prerequisite: though this kind of information is not generally made available in current practice, we believe that its importance will be increasingly recognized, since it enforces a sound discipline in design activity, besides being a useful tool for interaction analysis. The approach has been experimented through the implementation of the SIC prototype, applied to a simple interactive system. Though experimental results are very preliminary, they seem to indicate that the approach has an interesting potential.

As to future research, we plan the study of how to derive the control automaton from observation, of how to use plausible reasoning techniques in interaction pattern recognition, and of automatic or semi-automatic techniques for deriving proper modifications to the interactive system.

11. ACKNOWLEDGMENTS

The authors are indebted to the anonymous referees for their insightful comments. The support of grants by MURST 60%, and MURST Cofin 2000 is acknowledged.

12. REFERENCES

[1] Armour, P., The case for a new business model. Comm. Acm. 43(8), August 2000, 19-22.

[2] Barnard, P., May, J., Duke, D., Duce, D., Systems, Interactions and Macrotheory. Acm Trans. On Human-Computer Interaction, 7(2), June 2000, 222-262.

[3] Berners-Lee, T., What the Semantic Web can represent, 1998, http://www.w3.org/DesignIssues/RDFnot.html.

[4] Booch G., Rumbaugh, J., Jacobson, I., The Unified Modeling Language User Guide. Addison Wesley, Reading, MA, 1999.

[5] Bottoni, P., Costabile, M. F., Levialdi, S., Mussio, P., Defining Visual Languages for Interactive Computing. IEEE Trans. on Systems, Man, and Cybernetics, 27(6), 1997, 773-783.

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

[7] Bottoni, P., Costabile, M. F., Fogli, D., Levialdi, S., Mussio, P., Multilevel Modelling and Design of Visual Interactive Systems. Proc. of the IEEE Symposia on Human-Centric Computing Languages and Environments, Stresa, I, 2001, 256-263.

[8] Brown, J., Graham, T.C.N., Wright, T., 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.

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

[10] Cingil, I., Dogac, A., Azgin, A., A Broader Approach to Personalization. Comm. ACM, 43(8), 2000, 136-141.

[11] Dix, A. J., Formal Methods for Interactive Systems. Academic Press, 1991.

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

[13] Fogli, D., Mussio, P., A systemic approach to the specification and design of usable interactive systems, Internal Report, University of Brescia, 2001.

[14] Guzdial, M., Deriving software usage patterns from log files, Tech. Rep. GIT-GVU-93, 41, 1993.

[15] Harel, D., On visual formalisms. Comm. of the ACM, 31(5), 1988, 514-529.

[16] Hilbert, D. M., Robbins, J. E., Redmiles, D. F., 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.

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

[18] Hutchins, E.L, Hollan, J. D, Norman, D., Direct manipulation interfaces. In User Centred System Design, Norman D. and Draper S., eds., 87-124. Hillsdale, NJ: Lawrence Erlbaum Associates, 1986.

[19] Jennings, N., An agent-based approach for building complex software systems. Comm. of the ACM, 44(4), 2001, 35-41.

[20] Lelewer, D. A., Hirschberg, D. S., Data Compression. ACM Computing Surveys, 19(3), 1987, 261-296.

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

[22] Manber, U., Patel, A., Robison, J., Experience with personalization on Yahoo! Comm. of the ACM, 43(8), 2000, 35-39.

[23] Maulsby, D., Inductive Task Modeling for User Interface Customization. ACM Int. Conf. on Intelligent User Interfaces, Orlando, FL, USA, 1997, 233-236.

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

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

[26] Prates, R., De Souza, C., Barboza, S., A Method for Evaluating the Communicability of User Interfaces. Interactions, 7(1), 2000, 31-38.

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

[28] Riecken, D., Personalized views of personalization, Introduction to special issue on Personalization. Comm. of the ACM, 43(8), August 2000, 27-28.

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

[30] Spiliopoulou, M., Web Usage Mining for Web Site Evaluation. Comm. ACM, 43(8), 2000, 127-134.

[31] W3C Consortium, Extensible markup language (XML), 2001, http://www.w3.org/XML