Designing Synchronous User Interface for Collaborative Applications

Antonio Mendes da Silva Filho
Hans Kurt Edmund Liesenberg

Relatório Técnico IC–99-02

Janeiro de 1999

Designing Synchronous User Interface for Collaborative Applications*

Antonio Mendes da Silva Filho†
Hans Kurt E. Liesenberg‡
Institute of Computing
State University of Campinas
amendes|hans@dcc.unicamp.br

Abstract

Synchronous User interface is a medium where all objects being shared on it can be viewed indifferently from the geographical location and its users can interact with each other in real-time. Designing such an interface for users working collaboratively requires to deal with a number of issues. Herein, our concerns lies on the design of control component of Human-Computer Interaction (HCI) and corresponding User Interface (UI) software that implements it. We make use of our approach to interactive system development based on the MPX - Mapping from PAN (Protagonist Action Notation) into Xchart (eXtended Statechart) - and illustrate it by presenting the case study of a collaborative application.

Keywords: PAN, MPX, HCI design, UI software design.

1 INTRODUCTION

To survive, human beings need to organize themselves into a society. Differently from other animals that are able to live separately in reasonable manner, human beings are endowed with physical and cognitive abilities needed for attaining, by themselves, the living conditions. Human beings have survived and made progress because they have lived in groups and learned to divide tasks. By dividing tasks and using supporting tools, quality and productivity can be improved, and human beings empowered. All of this highlights some social aspects observed since remote human origins.

Today, we are living the information era. In this era, people can perform several activities without regard to geographical location. In that sense, it is possible to interact with colleagues, share data and computational resources, access intrumentation, and access information in digital libraries. This has become possible because of the big merge of this century involving the computer and communication industries. All these activities involve collaboration between different partners. For instance, research colleagues can work collaboratively in a Physics Collaboratory as reported by Agarwal et al [1]. Another example is that of a collaboratory based on virtual records which allows health practitioners to work toghether more effectively, delivering a higher quality care to people as reported by Kilman and Forslund [7]. We can have as well a Web-based collaborating system, such as the Digital Agora by Watters et al [19] where the system aims at providing support for active learning in Social Sciences. Users of that system are typically

* This research is partially supported by the Brazilian Council of Research (CAPES).

† Assistant Professor in the Department of Informatics at the State University of Maringá, Maringá, PR, Brazil, 87020-900. Email: amendes@din.uem.br.

‡ Director of the Computer Center at the State University of Campinas.

students, faculty and institution advisors. Other applications that can be considered are Digital Libraries [2, 13], and Web-based Entertainments like multiuser Web games [18].

The above examples illustrate scenarios where performers act collaboratively within a shared workspace. In the above mentioned examples, there are both a group of performers and a set of objects they are working on. Herein, objects are used in general sense. Within this context, we want a *synchronous user interface* in which all objects being shared on it can be viewed indifferently from the geographical location and the performers can interact with each other in real-time. In such a case, a collaborative software demands quick access to the objects for the purpose of rendering within a single view. Getting into this point involves the provision of support to collaborative systems design. The following sections address these issues. Our concerns lie mainly on the design control component of Human-Computer Interaction (HCI) and corresponding User Interface (UI) software that implements it. Two important features being addressed in this paper are the mode of interaction the system supports and the geographical distribution of users. The collaborative system presented in our case study addresses the interaction mode occurring *synchronously*, and the geographical *remote* distribution where users are at different locations. A further discussion on that is postponed to Section 4.

The following background issues on interactive systems development process are given. Our approach based on protagonist tasks for HCI design is presented in Section 3. Section 4 illustrates the use of MPX-based approach through the case study of a collaborative system where MPX is applied. Concluding remarks are presented in Section 5.

2 BACKGROUND ISSUES

Designing interactive systems involves a great deal of exploration of alternative solutions. Each solution has to be analyzed in order to evaluate its suitability. These activities lead in turn to a constant need for communication between all the people within the design. All these design issues depend on appropriate methods for describing the solutions at hand.

The development of large interactive systems is complex and it becomes even more difficult because of the need of involving the user as an active participant during the entire process. Henceforth, we need to have a continuous evaluation during the whole process as suggested by Hartson and Hix [5]. This requirement justifies the need of incorporating human factors into the system development aiming to achieve quality interfaces.

Besides this requirement, the HCI design has to be transformed into UI software design to ensure the correctness of the specification defined at a prior stage. In other words, we need to map the user-centered HCI design smoothly into system-centered UI software design. For this reason we need specific representation techniques and tools to support these two major stages of an interface development process. On the one hand, HCI involves user actions, interface feedback, screen appearance, and user tasks. On the other hand, UI software design involves algorithms, data structures, widgets, and calling structure of modules.

Within this context, we have been investigating how models of user tasks can contribute to the design and development of an interactive system. In that sense, our approach for developing interactive systems is based on MPX. This methodological support takes place by improving support for interactive system development life-cycle and by facilitating early exploration of design alternatives. A further discussion on interactive system development process is given by Hix and Hartson [6], pp. 103-115. Therein, we identify the two major stages of the process: the HCI design and the UI software design. Within this context, our concerns is focused on what and how representation techniques should be used at these stages and how information are gathered, refined and communicated to the other stages.

At the next section we start presenting early the stage of an interactive system development process by discussing our HCI design approach based on protagonists tasks. In that sense, our

work [16] shares the ideas pointed out by Grudin [4] that since most work takes place within a social context, computer systems are to consider the social and organizational knowledge in order to support them.

3 PROTAGONISTS TASK-BASED HCI DESIGN

In this Section an approach to capture HCI design is presented. Myers et al [12] defines Human-Computer Interaction (HCI) as *"the study of how people design, implement, and use interactive computer systems and how computers affect individuals, organizations, and society"*. Herein, our interest lies on the way people interact with computers, i.e. the control component of HCI design. In that sense, in the introductory section, we point out that the integration of the computer and telecommunication areas has re-shaped the traditional computing where most of users are used to work on stand-alone machines. Today the traditional computing is giving way to the social computing which allows an ever increasing interation of research fields, people of diverse backgrounds and abilities, and different cultures. Schuler [3] says that *social computing describes any type of computing application in which software serves as an intermediary or a focus for a social relation.* In that sense, a set of difficulties to design user interfaces has been pointed out by Myers [11]. They are:

- the difficulty in knowing tasks and users;
- the inherent complexity of tasks and applications;
- The variety of different aspects and requirements;
- theories and guidelines are not sufficient;
- difficulty of doing iterative design.

In order to tackle these afore-said difficulties and to better support the process of developing interactive systems we consider the tasks carried out by protagonists. Protagonists are all the components that play roles in an interaction scenario. Such protagonists are both user(s) and system components. These components can as well be viewed as software components with major functionality roles. Herein, we are concerned with the tasks carried out by protagonists. In this respect, human beings often use computers to carry out their tasks. We can have tasks as guiding elements for HCI design.

This approach for HCI design is proposed due to the diversity of users with different knowledge levels as well as a variety of both interaction styles and implementation technology. This need becomes even greater in social computing scenarios discussed by Silva Filho and Liesenberg [16] due to the inherent complexity.

We also advocate the need to start protagonist task-based HCI design at a high abstraction level where only user intentions are captured. This makes the HCI design easier for handling and upgrading as the refinement process proceeds until detailed requirements are obtained and implementation decisions are made. Besides the use of the protagonist task-based approach, we can get users involved early enough in the design process provided that appropriate metaphors related to their daily tasks are used. Using metaphors in such a process augments the integration between users and designers because they can communicate effectively between each other using a common language derived from such metaphors. A HCI design based on metaphors of the domain aims at facilitating the user involvement in the design process.

User intentions at the task abstraction level are high-level goals which exist in the user's conceptual model about the system. User intentions to reach a goal are described at a high abstraction level without reference to any system presentation feature. For instance, in a file

system *delete a file* represents a user intention in the task abstraction level. However, *drag a file icon to a destination* (e.g. trashcan icon), type a command *rm file* or utter the command *remove file* are respectively descriptions of low-level user intentions with desktop, command-based, and natural language interfaces that perform the associated high-level intention. Below we present the steps needed for our approach.

1. *System description* - Obtain a system description of the system under development (SUD).
2. *System protagonists identification* - These protagonists can be both user and system components depending on decisions of what parts of a working process should or should not be carried out automatically. To do so, we use the Protagonist Action Notation (PAN) [15].
3. *Development of interaction scenarios involving system protagonists* - For every protagonist, the designer must describe scenarios in order to identify its tasks.
4. *System architectural model identification* - Herein, we seek a system description in terms of its architectural model.
5. *System task representation* - In this step, we define a task set for each protagonist by using Task Coordination Models (TCMs).

So far we have presented the main steps of our protagonist task-based HCI design approach. To support it, we illustrate in the next Section how to carry out the HCI design through an example of collaborative application. Before going on, it is worth observing that we can have different collaborative relationships as given in Figure 1.

Collaborative relationship	Collaboration level	Knowledge about other partners	System complexity
stand-alone component	none	no	low
mediator	medium	no	medium
Everyone knows each other(EKEO)	high	yes	high
Mediator and EKEO(MEKEO)	medium	yes/no	high

Figure 1: Board of DirectionLeader.

A collaborative relationship describes how the partners or components of a system interact with each other. A set of partners can collaborate with each other in such a way that they can observe changes of states and/or data values from other partners. A stand-alone component does not interact with other partner in the system. In general, its state and operation does not affect overall timing constraints. For instance, a power display component is only receiving the power reading and showing the power value without interacting with any other partners.

On the other hand, interactions among different partners can be coordinated through a mediator. In this case, every partner registers the services which it can provide and the services needed from other partners while the mediator keeps records of provider and requester profiles.

Another collaboration pattern is everyone knows each other (EKEO) which allows partners to register and request directly one from another. In that case, each partner has to know

the interfaces of the other partners and requests the service(s) through these interfaces. We may as well have a mediator acting as a *broadcaster* and, therefore, not interfering with the communication process among partners. In fact, what we have is a *peer-to-peer* architecture where all the partners involved in a collaboration relationship can be either serving or clienting at any time. We get back to that when we present the case study.

Finally, we have another collaboration pattern where there exists a mediator and everyone knows each other (MEKEO). In this case, partners can request services either from the mediator or directly from providers through their interfaces.

4 CASE STUDY OF A COLLABORATIVE APPLICATION

This Section presents an example of collaborative application, called *DirectionLeader* - a multiplayer game - which involves multiple users at different locations. We start providing background issues on multiplayer games and game theory to underlie the presentation of this case study and then we illustrate the use of our approach to describe interaction aspects of DirectionLeader.

4.1 Multiplayer Games

Multiplayer Games that can be run on multiple machines are also known as *Network Games*, which means that the games are capable of enabling multiple players to play interactively on top of a network. In the case of networked Web games, the communication between players is mediated by the Internet. In other words, in a networked Web game which involves two or more players, players are able to play the game together and interact with each other via their Web connection in a concurrent manner.

In multiuser games, the communication design can be affected by the way the game play progresses, which is determined by the type of the particular game. Most games fall into one of two categories:

- *Turn-based games* are games in which each action in the game is based on a player's turn;
- *Event-based games* are games that are paced by input events that can occur at any time.

DirectionLeader is a event-based game because players can make a move at any time. Some issues of game theory are given in the following subsection to underlie the case study presentation.

4.2 Game Theory

Game Theory is a research area devoted to the study of decision making in conflict situations. It can be used to shed light on how people interact with each other in a multiplayer computer game scenario. Game theory might help a designer to figure out more creative approaches to the game strategy itself. Such a situation exists when two or more decision makers, or players, with differing objectives act on the same system or share the same resources. Game theory provides an underlying process for selecting an optimum strategy which depends on the moves of the opponents who have a strategy of their own. In game theory, the following assumptions are usually made:

- each player has two or more well-specified choices or sequences of choices called moves;
- every possible combination of moves available to the players leads to a well-defined end state (win, loss, tie, or draw) that terminates the game;
- a specified payoff for each player is associated with each end state;

- each decision maker has perfect knowledge of the game and of their opponents, i.e., he knows in full detail the rules of the game as well as payoffs for all other players;
- all decision makers are rational, i.e., each player, given two alternatives, will select one that yields the greater payoff.

Although general in scope and not originally directed at computer games, game theory touches on many of the same concerns that are raised when strategies for multiplayer computer games are being designed. Two players in a network multiplayer game often go through much of the same thought pattern as people engaged in a verbal conflict. Game theory applies equally to both scenarios. More details on game theory are given by Russel [14] and Luger [10].

4.3 Designing DirectionLeader

DirectionLeader is a multiuser game where the major goal is to lead the direction of a set of followers that can go to any direction under the command of the leader. The following presents the full game description and we use our approach based on protagonist tasks to carry out the HCI design.

4.3.1 System Description

The game description is as follows. An illustration of the screen appearance is shown in Figure 2.

> DirectionLeader is played with a rectangular board that may contain up to 64 followers on the workspace. Followers appear on the screen as small balls at different positions which identify each participating player and their direction of movement are guided by a leader. That is, the leader is protagonized by the player that can change the follower movement direction at his/her will. Thus, each leader can recruit their followers by making a selection out of 64 available followers on the system. They are identified with a simple binary string and a selection can be made by pressing buttons until the desired number of followers is attained (see Figure 2). For example, consider a fictitious leader named *Bill* with four followers given by the binary string *00*1*0*. In that case, the followers are: *000100*, *001100*, *000110*, and *001110*. The asterisk symbol is used as a wildcard. Note that DirectionLeader is a event-based game because every player can make a move at any time. DirectionLeader can be played with any number of players. Each player has its associated window/machine where the game is played, that can run on the machine of the server or can be run from other locations. Every follower has a screen location and a direction. When a player clicks near a follower, it sprouts a little rod and prepares to move in the direction from where the leading player clicked based on the moves motivated by the server. The speed at which your follower moves will be inversely related to the number of existing followers you have moving at any time. It is worth observing that the followers you see at your screen are as well seen by other players at different locations, i.e., the workspace are shared in real-time among all the involved players. Furthermore, when your mouse click has affected a follower, the associated name attribute changes and promptly shows name of the related player. While players keep making their moves, a server listens to changes in the direction attribute and when there is a direction available, it moves to a new location according to the direction with each tick.

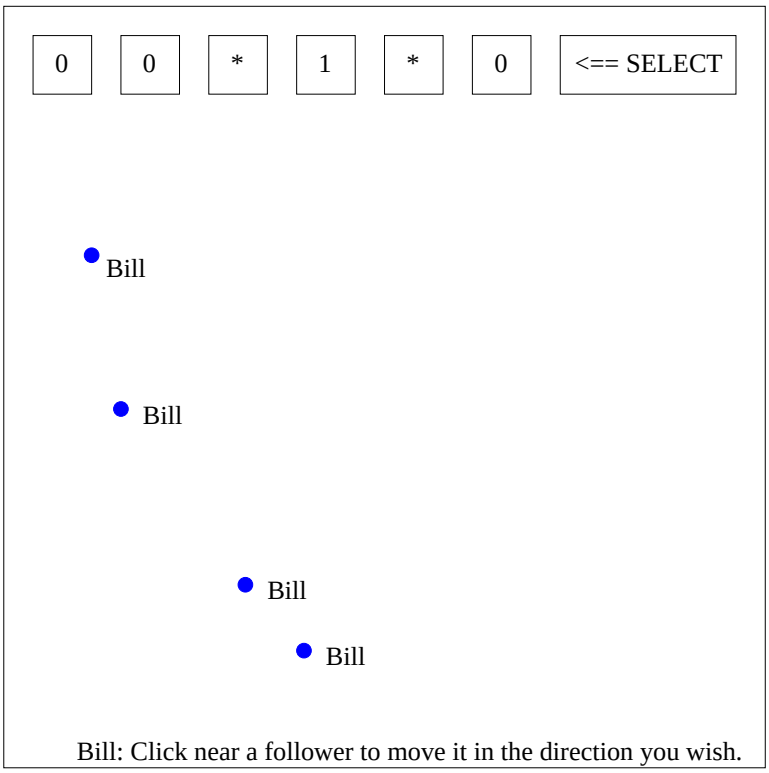

Figure 2: Board of DirectionLeader.

After obtaining an system description as above, we need to identify system protagonists as shown below. We present as well system components and illustrate the architectural model of DirectionLeader.

4.3.2 Identification of System Protagonists

Protagonists are all the components that play roles in an interaction scenario. Such protagonists are both user(s) and system (or system components). These components can be viewed as software components with major functionality roles. In order to carry out tasks, interactions between these protagonists must occur. In that case, we need to capture not only user actions but also system-generated actions. Thus using a notation that captures only user actions constitutes a hindrance to the user interface design needs. Consequently, a designer needs a notation that allows him/her to capture the whole interaction picture. In our example, each player interacts with the corresponding player interface component and notifies his move to all other players involved in the game. Note that a player is protagonized by a user generally at a different location. When a move is made, it is sent out to coordinator that works in background listening to the moves and propagating them to all other players. Note that propagation only takes place when they occur which is a nicer solution differently of a polling approach that causes lots of unnecessary network traffic. Figure 3 shows the protagonists involved in this system. Note that we may have any number of players interacting with each other and one coordinator that does not interfere with the interaction among all participants. The coordinator does the broadcasting of all player moves maintaining the syncronized replicas of the same workspace on each site. In this case, everyone can see an up-to-date screen view in real-time. This collaboration type (EKEO) is the one which requires more from a designer as presented in Figure 1 (see Section 3). To support quick access to the players for the purpose of rendering within a single view, the status of the players are cached so that a *repaint* cause no network traffic.

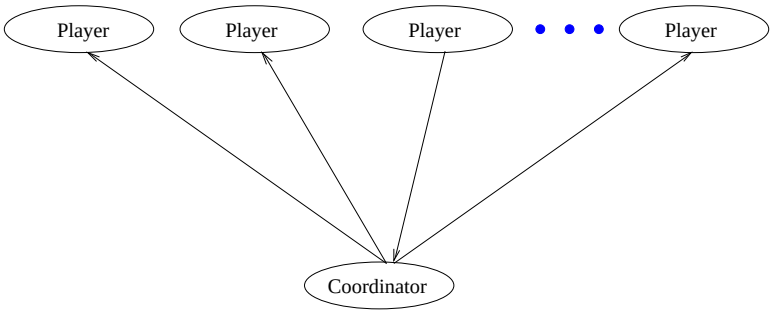

Figure 3: DirectionLeader protagonists.

Figure 3 shows the protagonists identified for DirectionLeader. To do that, we use PAN (Protagonist Action Notation) to provide designer with a high abstraction level notation that allows him/her to document the HCI requirements. Another use of PAN is reported by Silva Filho and Liesenberg is given in [15] for in other collaborative application.

4.3.3 Development of Interaction Scenarios

Let us make an assumption that, at a certain time, there are only three players, named Bill, Jack, and Bob for the purpose of illustration. Each protagonist is capable of performing particular tasks as given below. Nevertheless, before we present task descriptions for each protagonist we identify an architectural model for the collaborative system. A way of doing that is developing interaction scenarios involving protagonists. At a high abstraction level we can characterize an interaction scenario in DirectionLeader as shown in Figure 4. Therein, one of players, called *Jack* changes the *direction for followers* as e.g. *go to Northeast*, and this player's move is propagated by the coordinator to all other players (Bill and Jack) through the *player update* action. Additionally, we abstract from all interaction details such as *the kind of information exchanged between protagonists*. We focus on interactions between protagonists. Later on we deal, more specifically, with protagonist actions.

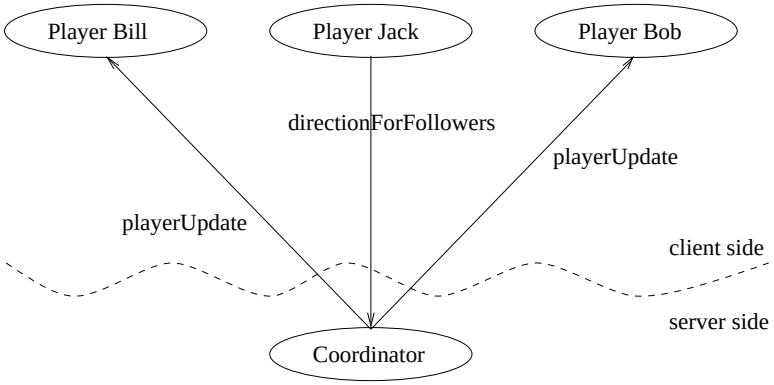

Figure 4: Scenario with application protagonists.

4.3.4 Identification of the System Architectural Model

At this step, we do a system description in terms of their protagonists. It is worth pointing out that both users and software components can play the role of protagonists. Figure 4 give us an idea of a possible architectural model for the system. It is worth rethinking the client/server model. The key difference between server and client in a collaborative system is their one-to-many relationship, like a *broadcasting* between a single peer and the multiple remote peers. We

called such an architecture as *peer-to-peer* since it involves peer components that can either be a server or a client at any time. Nevertheless, note that all of these peer components always see an up-to-date view of the shared workspace (i.e., the game screen). To do so, a form of *caching* is used in that synchronized full replicas of the peers are maintained on each client-side. So we identify the *peer-to-peer* architetural model for this system.

4.3.5 Task Description of DirectionLeader

From the identification of the protagonists and their interactions at a high abstraction level an architectural model for DirectionLeader has been identified. Such identification was derived smoothly based on the used interaction scenario where each protagonist (players and coordinator) is capable of performing some particular tasks. Below we present the Coordinator tasks.

Tasks of the Coordinator:

- wait for player updates or coordinator-initiated actions;
- receive update notification from player and propagate it to all other players in the game;
- broadcast coordinator-iniated actions to all players;
- terminate the game upon request.

Figure 5 illustrates the Task Coordination Model (TCM for short) for the coordinator. It is worth observing that Figure 5 not only shows the tasks of the coordinador but reveals as well the existing relationship among them, i.e., how the task coordination takes place. The particular TCM of Figure 5 represents the tasks and their relationships of just one protagonist given in the PAN. In other words, it describes tasks of and how they are coordinated by the coordinator protagonist.

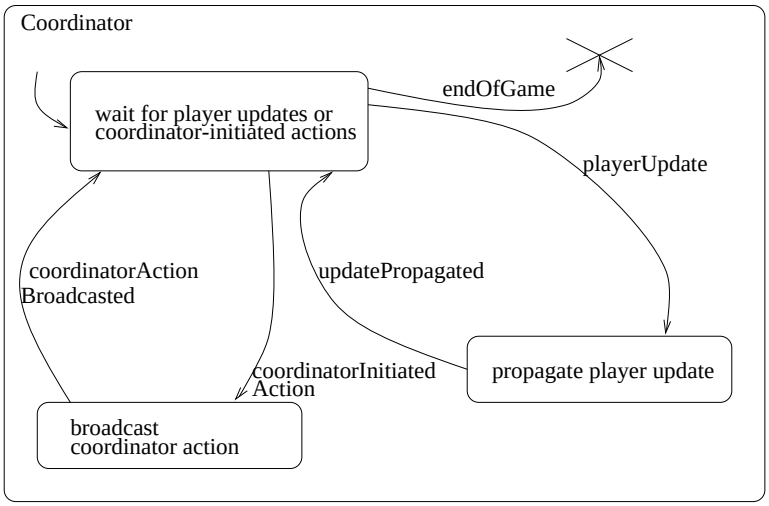

Figure 5: TCM for coordinator.

The reader should note that the designer does not conceive this model on the fly. Initially, he identifies the tasks, then how tasks are related to each other, and finally what stimuli cause navigation between tasks. System's users are also involved within this process as reported by Liesenberg and Silva Filho [16]. Furthermore, Figure 5 illustrates that the termination of a task can motivate a navigation to another one or actions of other protagonists may as well cause

task navigation in the TCM of the coordinator as when it receives a request from a player for updating the game status.

Directed edges in TCM are labelled with stimuli caused by events generated by protagonists. An event can be passed from one protagonist to another one. Additionally, events can carry data between protagonists. The reader should as well note that a coordinator does not interfere in a game. It is in charge of mainly detecting player updates and propagates them to other players.

The other DirectionLeader protagonist is a player. A player component represents a user interface module which receives actions from its associated user and maps them into moves. The tasks of a player is given below.

Tasks of a player:

- at the beginning, register on the coordinator;
- after registered, make a selection out of the 64 available followers on the coordinator;
- once the number of followers is selected, choose the direction the followers are to go, perform a direction change, and inform the coordinator about it;
- at any time, a player can change either the number of followers or their direction.

Note that the main task of the players is to inform the coordinator about their moves that can either be a follower number selection or a follower direction change. Moreover, a player needs to reason about the game in order to carry out a new move. Figure 6 gives the TCM for a player.

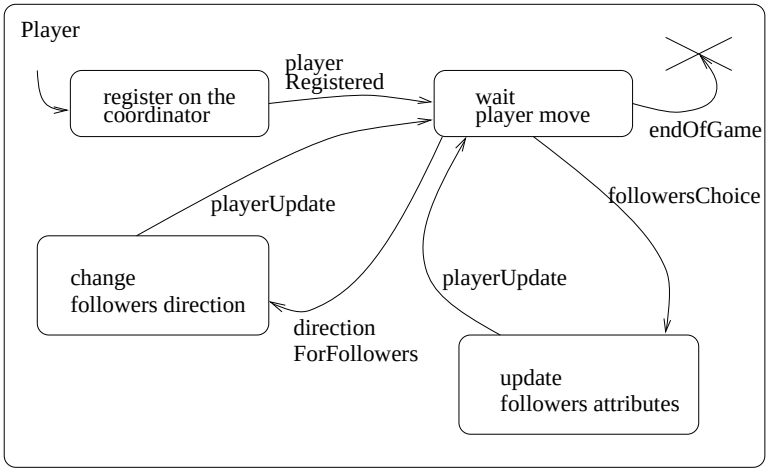

Figure 6: TCM for player.

4.4 MPX - Mapping PAN into Xchart

As pointed out in Section 2, the process of developing interactive systems consists of developing both noninterface software (functionality) and interface software. Herein, our interest lies on the latter one. We as well discussed that the two major phases of interface software development process are HCI design and UI software design. Above, we have presented our approach based on protagonist tasks for HCI design. At this point we are concerned with the process of mapping HCI design into UI software design. Reasons for doing that were discussed earlier in this paper.

The goal of protagonist task-based HCI design is not strictly ease of design, but rather capturing the way the user sees the dialogue. The idea is to capture the user's view of an interface. This is carried out by describing the dialogue between the system components and the user (i.e. dialogue protagonists of system under development), as seen by the user, rather than to describe the structure of the system of its components at some level. The ultimate goal is that users see the dialogue as a collection of protagonists communicating with each other. With that in mind, Figure 7 illustrates the stages of our approach.

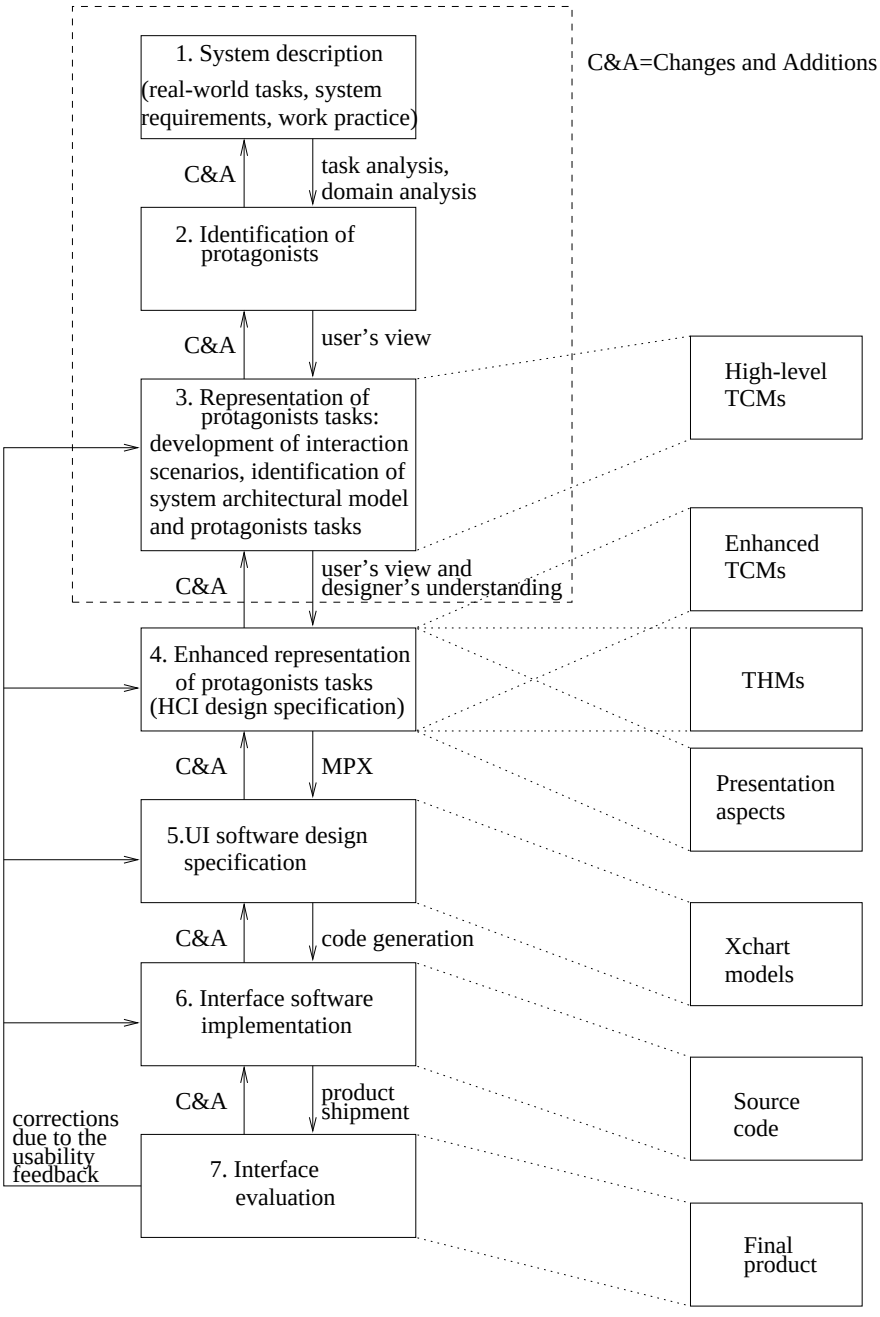

Figure 7: Interactive Systems Development using MPX.

The upper part in the dotted line shows HCI design based on the protagonist tasks as presented in subsection 4.3. For example, consider a collaborative application example worked out in this paper. In developing a HCI design for collaborative application, it is of paramount importance to consider that most system usability comes from the system organization. This

reflects the model of the user. In other words, the embedded model of the user and the extent to which the user can understand the (organizational) model provided by the system and the way he/she can interact with the system based on his/her understanding.

The reader should note that designer does not obtain TCMs for protagonists on the fly. Initially, after identifying protagonists, interaction scenarios are developed together with users which aims at getting a picture of the way protagonists communicate with each other. These interaction scenarios help identifying both the system architectural model and the protagonist tasks. This can be carried out by observing users at their workplace and also by an interviewing process where an initial set of tasks are brought out. This is carried out in order to gather the main tasks for every protagonist. In this context, user and designer play the roles of collaborators. With the task description attained by means of this process and with the support of interaction scenarios developed at an earlier stage, the designer can represent tasks and navigation between them by using TCMs. The designer identifies as well what stimuli cause navigation between tasks. Elaboration of high-level TCMs are performed with user participation. At the end of stage 3 we obtain *high-level TCMs* as shown on the right-hand side of Figure 7. Furthermore, the system architectural model identified at this stage is as well derived from the developed interaction scenarios. These interaction scenarios help to identify the existing interaction pattern among the protagonists and, henceforth, the related architectural model for the system at hand. Note that these TCMs are high-level and we have the need for a further refinement before performing the mapping into UI software design specification. The enhanced TCMs and THMs (Task Hierarchical Model) are given in [17] (because of limitation of space). Enhanced TCMs are a combined and detailed views of users and designer. Note that TCMs, THMs and presentation aspects are simply distinct viewpoints of the same entity, that is a set of tasks. TCMs illustrate the way tasks are coordinated by a protagonist while THMs shows the how tasks are hierarchically organized and presentation aspects deal with interaction objects that allow users enter command and/or data and the way results are displayed. A further discussion can be found in [17].

With the enhanced representation of tasks at hand, MPX supports an automatic generation of UI software design specification given by the XChart Model (XCM) [9, 8]. After obtaining the Xchart model, source code for an interface can be automatically generated, compiled and executed. Further details about Xchart language and environment can be found at: http://www.dcc.unicamp.br/proj-xchart/

Once stage 4 has been finished, we are in position to pursue the mapping from HCI design into UI software design. MPX does that automatically. MPX is an algorithm which takes HCI design specification (that contains dialogue patterns of each protagonist) as input and generates the respective UI software design. The output is a set of Xcharts which reflect the HCI design performed earlier. The Xcharts for coordinator and player of the collaborative application we have been working on are given in Figures 8 and 9, respectively.

It is worth observing that during design and implementation a set of requirements were worked out, such as: everyone should receive a quick notification when a partner's attribute changes, update notification should be based on listener, and network traffic should be minimized by using a strategy of caching. Within this context, the collaborative application has a one-to-many relationship where a single player propagates its move to other partners.

In stage 7, we are allowed to carry out evaluation with a running prototype. Note that we can perform changes at different stages which aims at making either addition(s) or change(s). However, this topic is out of the scope of this paper.

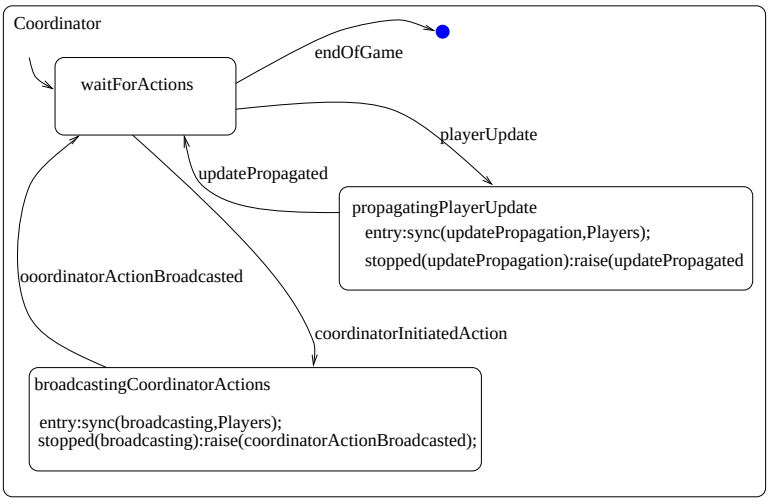

Figure 8: Xchart for Coordinator.

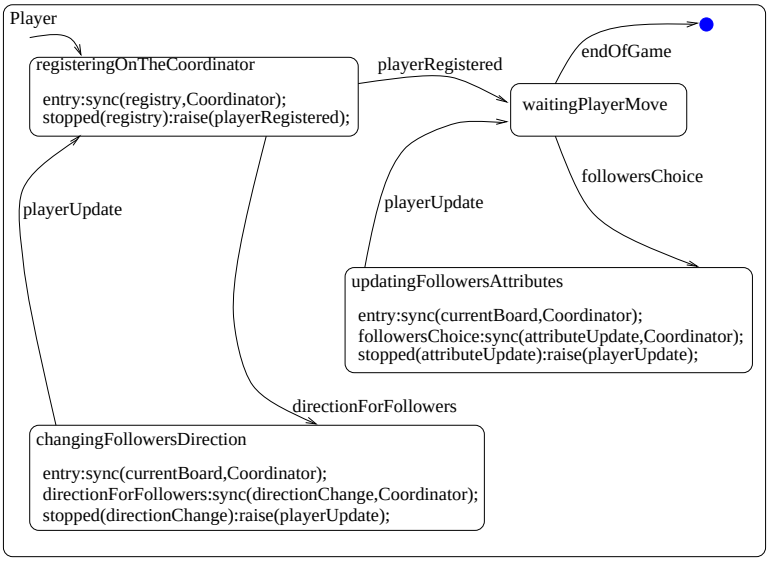

Figure 9: Xchart for Player.

5 CONCLUDING REMARKS

This paper has presented a synchronous user interface design for a collaborative application. Herein, we focused on the design of the control component. A case study involving a collaborative application has been worked out where the main difficulty comes from the fact that we need provide its users with a synchronous user interface. Such an interface acts as a medium where all objects being shared on it can be viewed independently from the geographical location and its users can interact with each other in real-time. The two major stages in the interactive system development process, i.e. HCI and UI software design, were carried out by using PAN and Xchart, respectively. In that case, we have used our methodological approach based on MPX where we map the user-centered HCI design smoothly into the system-centered UI software design. As a result of this mapping, the development process becomes more effective, development time is reduced, expertise can be used in specific stages of development process, time for market is minimized, and user interface quality improved. Next step of this research is the development of graphical tools to support development process.

6 ACKNOWLEDGEMENTS

The authors would like to thank the financial support from CAPES.

References

[1] D. A. Agarwal, S. R. Sachs, and W. E. Johnston. The Reality of Collaboratories. *Computer Physics Communications*, 110(1–3):134–141, May 1998.

[2] IEEE Computer. Special Issue on Digital Libraries Initiatives. *IEEE Computer*, 29(5), May 1996.

[3] D. Schuler (Guest Editor). Special Section on Social Computing. *Communications of the ACM*, 37(1):28–80, Jan 1994.

[4] J. Grudin. The Computer Reaches Out: The Historical Continuity of Interface Design. *CHI'90 Conference Proceedings, In Empowering People (J. C. Chew and J. Whiteside, eds.)*, pages 261–268, 1990.

[5] H. R. Hartson and D. Hix. Toward Empirically Derived Methodologies and Tools for Human-Computer Interface Development. *International Journal of Man-Machine Studies*, 31:477–494, 1989.

[6] D. Hix and H. R. Hartson. Developing User Interfaces: Ensuring Usability Through Product and Process. *Reading*, 1993.

[7] D. Kilman and D. W. Forslund. An International Collaboratory Based on Virtual Patient Records. *Communications of the ACM*, 40(8):110–117, Aug 1997.

[8] F. N. Lucena. XCHART: A Model for Specifying and Implementing Dialogue Managers. *PhD Thesis, Institute of Computing, State University of Campinas, Brasil*, Dec 1997.

[9] F. N. Lucena, M. M. Harada, and H. K. E. Liesenberg. Operational Semantics of Extended Statecharts (XCHART). *Proceedings of the 3rd Workshop on Logic, Language, Information and Computation, Brasil*, May 1996.

[10] G. F. Luger and W. A. Stubblefield. Artificial Intelligence: Structures and Strategies for Complex Problem Solving. *(Reading), The Benjamin-Cummings Publishing Company, Inc.*, 1993.

[11] B. A. Myers. Why are Human-Computer Interfaces Difficult to Design and Implement? *Technical Report CMU-CS-93-183, Computer Science Department, Carnegie Mellon University*, July 1993.

[12] B. A. Myers and et al. Strategic Directions in Human-Computer Interaction. *ACM Computing Surveys*, 28(4):794–809, Dec 1996.

[13] Comm. of the ACM. Special Issue on Digital Library. *Communications of the ACM*, 41(4), April 1998.

[14] S. Russel and P. Norvig. Artificial Intelligence: A Modern Approach. *(Reading), Prentice Hall*, 1995.

[15] A. M. Silva Filho and H. K. E. Liesenberg. Capturing Computer-Human Interaction Design via the Protagonist Action Notation. *Proceedings of the 18th Brazilian Computer Society Annual Conference, Belo Horizonte, MG, Brazil*, 1:276–296, Aug 1998.

[16] A. M. Silva Filho and H. K. E. Liesenberg. Gathering User Interface Design Requirements for Social Computing. *Submitted to International Journal of Human-Computer Studies (a earlier version is given in the Technical Report IC-98-38, Institute of Computing, State University of Campinas, Brazil)*, Oct 1998.

[17] A. M. Silva Filho and H. K. E. Liesenberg. Toward Deriving User Interface Software Design From Human-Computer Interaction. *Technical Report (in preparation), Institute of Computing, State University of Campinas, Brazil, http://www.dcc.unicamp.br/proj-xchart/*, 1999.

[18] Yahoo Web Site. Interactive Web Games/Multi-User Games. *Available at <http://www.yahoo.com/Recreation/Games/Internet_Games/Interactive_Web_Games/Multi_User_Games/>.*, 1998.

[19] C. Watters, M. Conley, and C. Alexander. The Digital Agora: Using Technology for Learning in the Social Sciences. *Communications of the ACM*, 41(1):50–57, Jan 1998.