User Interface Design for Web Collaborative Systems

Antonio M. da Silva Filho
Department of Informatics
University of Maringa
amendes@din.uem.br

Roberto S. M. de Barros
Center of Informatics
University of Pernambuco
roberto@cin.ufpe.br

Hans K. E. Liesenberg
Institute of Computing
University of Campinas
hans@ic.unicamp.br

Abstract

Design for cooperation is a challenge. As designers we note that as we are getting into a new century, several areas have achieved significant breakthroughs. Among them, it is easy to perceive that Computing and Telecommunications have had an impact of paramount importance to society as a whole. These technologies have allowed an increasing integration of research fields, people of various backgrounds and abilities as well as made the interaction of different cultures possible. As a result, we have been living in the Internet era with a very large number of Web sites which can be visited, queried and played with. Application examples are Digital Libraries, Health Care Information Systems, Physics Collaboratories, and Web-based entertainments like interactive Web games. Within this context, we are concerned with the user interface design for such systems and a protagonist-oriented approach for capturing the user interface design is presented.

1 Introduction

The final years of the 20th century have led the whole society to a *digital maelstrom* due to significant breakthroughs achieved in the areas of Computing and Telecommunications. As time goes on, we are faced with the convergence of these technologies. Such an integration has made the reshaping of traditional computing possible where most users were used to work on stand-alone machines. Today the traditional computing is giving way to the *social computing* which allows an ever increasing interaction of research fields, involving people of various backgrounds and abilities as well as of different cultures. Social computing is one of the examples of the Internet era where a very large number of Web sites can be visited, queried and played with. Schuler [15] says that *"social computing describes any type of computing application in which software serves as an intermediary or a focus for a social relation"*.

We have identified a number of collaborative applications such as: Digital Libraries [4, 12], Health Care Information Systems [11], Physics Collaboratories [1], and Web-based Entertainments like multiuser Web games. We call them Web collaborative systems since it allows different players at different sites to play together in either a health care information system or a networked Web application. Another example of Web-based collaborative system is the Digital Agora by Watters et al[22], where the system aims

at providing support for active learning in Social Sciences. These examples illustrate scenarios where performers act cooperatively within a shared working environment.

Nevertheless, the popularity of the Internet has demanded more and more from interactive systems designers in the sense that they are concerned with the improvement of the usability of such systems. These networked systems have specific usability problems not found earlier on stand-alone machines. Internet browsers hide details of the underlying networks from the user. This leads to unpredictability because without that kind of information it is difficult to determine whether retrieval commands will be successful. For example, a remote site failure will prevent the requested information from being delivered to the user. Additionally, communication bottlenecks can delay the data transmission between a browser and remote servers. Within this context, we are concerned with the way designers can capture user interface (UI) design. To tackle this kind of problems, we present a protagonist-oriented analysis and design approach in order to capture the user interface designs.

Background issues on analysis and design approaches for social computing are discussed next. Section 3 presents the protagonist-oriented modeling technique for capturing the user interface design. A case study of a Web collaborative system using our protagonist-oriented approach is presented in Section 4 and concluding remarks are given in Section 5.

2 Background Issues

Computer systems are intended to aid people to perform their work. Henceforth, such systems must be built to satisfy the needs of their users. However, there is no way that a system can work well with a user group (e.g. a cooperative group) without a deep understanding of the users in such a group. Computer technology can and has acted as a direct aid to people at their tasks carried out during work activities. In addition, computer has changed dramatically the patterns of work and communication in a workplace. Note that computer technology alone cannot provide the answers when we deal with human activities. The tasks, the culture, the social structure, and the individual human beings are all essential components of the job and, unless the computational tools fit seamlessly in the structure, the result may be failure.

As designers, we need to take all these issues into account. Together with these observations we note that the transformation of telecommunications, brought about mainly by a close relationship with computers, is both driving down the cost of communication and driving up the amount of information that can be exchanged. In this context, we present a way of how designers can capture the user interface design for social computing which includes those systems mentioned earlier. Firstly, however, we discussed the two approaches which have inspired our proposal and, therefore, might also be alternatives in the design process. They are: Participatory Design and Ethnography-based Design.

2.1 Participatory Design

Participatory Design (PD) is an approach where future users of computer systems participate directly with designers in the design process. This approach, pioneered in Scandinavia, is widely accepted throughout Europe and has got attention in the USA. Clement

and Besselaar [3] give a historical review of the PD approach and identify the ingredients of PD projects: (i) Access to relevant information; (ii) Independent voice in decision making; (iii) User-controlled resources: time, facilities, expertise; (iv) Appropriate development methods; (v) Organizational/technical flexibility.

On top of the aforementioned issues, Greenbaum [6] discusses three reasons for the need of PD: from a *pragmatic* perspective, a *theoretical* perspective, and a *political* perspective. In the pragmatic perspective, she states that *"... it is generally acknowledged that approximately 60 to 80% of all problems can be traced to poor or inadequate requirement specifications. Obviously, computer systems need to better suit people's working practices"*. Now, from a theoretical perspective, she says that *"since systems developers and people at workplaces do not experience the same things, this limits how well they can understand each other's experiences. One way of getting around this dilemma is to apply a PD approach to prototyping which emphasizes providing people with hands-on experience in a work-like setting"*. While from a political perspective, she argues that *"As systems developers we have the obligation to provide people with the opportunity to influence their own lives. We believe it is our professional responsability not only to build systems that are cost-effective but that also improve the quality of work live"*.

2.2 Ethnography-based Design

Ethnography is a technique originally developed by anthropologists where they spend long periods of time in foreign societies aiming to understand the social mechanisms. Within a context of design, the major objective is to achieve a thorough understanding of work practices in order to better support the computer systems design.

This is an approach that has been receiving an increasing interest. By having its starting point anchored in the Social Sciences and the Humanities, it brings a provoking and relevant perspective into design. The focus is on the detailed analysis of current work practices, as viewed by the people who actually do the work. According to Blomberg et al [2], the four main principles that guide the ethnographic work are:

- *First hand encounters*: a commitment to study the activities of people in their everyday settings.
- *Holism*: a belief that particular behaviors can only be understood in the everyday context in which they occur.
- *Descriptive rather than prescriptive*: describe how people actually behave, instead of how they ought to behave.
- *Members' point-of-view*: describe the behavior in terms relevant and meaningful to the study participants.

The afore-said principles entail that designers should not pre-define any conceptual framework. Instead, designers are expected to capture the social structure in order to better support the cooperative work. Moreover, since work is a socially organized activity, where the actual behavior may differ from the way it is described by whoever does it, it is mandatory to rely not only on interviews but also on observations of everyday activities at the workplace where technology is supposed to be inserted. In that sense, it is similar to the Contextual Design approach (CD) of Holtzblatt and Beyer [7, 8] where design data

are gathered by having designers watching people doing their own jobs, interspersing observation, discussion, and reconstruction of past events.

3 Protagonist-oriented Analysis and Design

Protagonist-oriented analysis and design (PAD) is an approach based on the protagonist action notation(PAN) to capture the user interface design [19, 20, 16, 18], as shown in Figure 1.

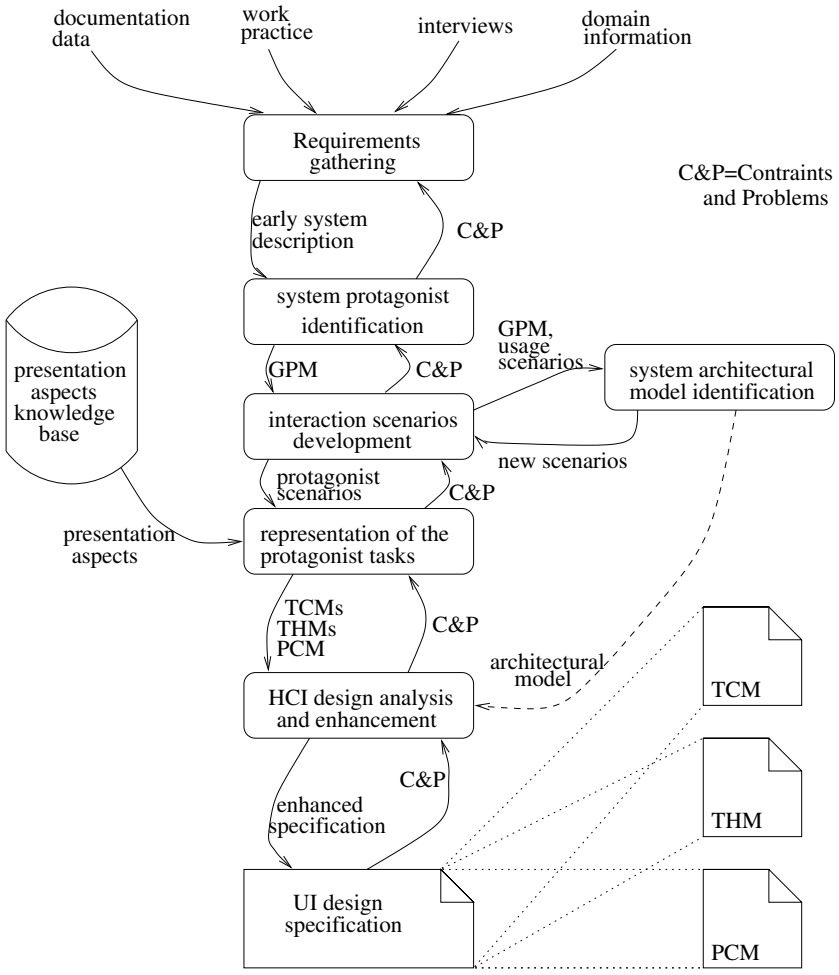

Figure 1: Protagonist-oriented Analysis and Design.

Requirements Gathering - It is concerned with understanding needs. That is, this phase aims at understanding the nature of the system to be developed and the required functionality. Within this context, the main objective is to capture the system's organization. In more detail, viewing such an organization as a collection of protagonists. In addition, this process aims at identifying the role that each protagonist will play. At the end, the designer is expected to obtain a system description of the system under development (SUD).

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. The designer makes use of the GPM to capture the system protagonists playing roles in interaction scenarios. A way of doing that is developing interaction scenarios involving protagonists.

Interaction Scenarios Development - The designer must describe interaction scenarios involving the system protagonists in order to identify their goals and tasks. It is worth observing that all interaction details such as *the kind of information exchanged between protagonists* are abstracted out. The focus is on the interactions between protagonists.

System Architectural Model Identification - A system is described in terms of its architectural model, i.e. the system is in terms of its components or protagonists. It is worth pointing out that both users and software components can play the role of protagonists.

System Protagonists Task Representation - From the identification of the protagonists and their interactions at a high abstraction level an architectural model is identified. Such identification is derived based on the interaction scenarios worked out where each system protagonist is capable of performing some particular tasks. In that case, the task set for each protagonist is described by using THMs, TCMs, and PCMs.

UI design analysis and enhancement - With the PAN, three models *task hierarchical model* (THM), *task coordination model* (TCM) and *presentation component model* (PCM) are used to obtain the UI design. Nevertheless, such models need to be analyzed and enhanced before generating the UI design specification. Note that all the models are given at a high abstraction level and so a further refinement is needed before mapping the UI design specification into UI software design specification [18]. With that in mind, an analysis and enhancement of the UI design is performed where users' and designer's views are combined and detailed. As a result, PAN models are enhanced to generate the UI design specification. It is worth observing that THM, TCM and PCM are simply distinct viewpoints of the same entity, that is the task set of the system protagonists. The main steps of our protagonist-oriented design approach for Web collaborative systems has been presented above.

3.1 Protagonist Action Notation

The primary abstraction of the PAN is the protagonist. Protagonists can be either a user or system components playing major roles in interaction scenarios. The primary abstraction of the PAN is the protagonist. An interactive system is viewed as a collection of protagonists interacting with each other, represented by a GPM. Each protagonist can perform one or more tasks. These tasks and their relationships are described through the TCM. Additionally, the structure of tasks is captured via the THM. The PAN is supplemented with the PCM, highlighting the presentation aspects, which provides the interaction objects (independent of the toolkits) to be used by the dialogue control component. Figure 2 shows the models that comprises the PAN. These models are used to capture the user interface design.

In addition, scenarios are used to better identify how a protagonist is related to another and to capture the sequencing among the tasks for each protagonist. It is worth observing that protagonists represent an advance in terms of abstraction, i.e. they may be used by software engineers to more naturally understand, model, and develop the important class of interactive systems. Note this section is concerned with the models and resources being used with PAD. They are TCM, THM and PCM. These models are derived from the GPM and are part of the PAN. The models are presented in the following subsections.

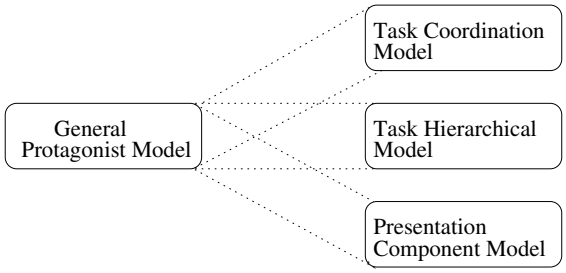

Figure 2: Models of the PAN.

3.1.1 General Protagonist Model (GPM)

GPM is aimed at capturing the protagonists of a SUD. Nevertheless, the *requirements gathering* phase precedes the modeling of protagonists as shown in Figure 1. Herein, we aim at capturing the organization of the system. Specifically, we view such an organization as a collection of protagonists. In addition, this process aims at identifying the role played by each protagonist. At the end, the designer is expected to obtain a system description of the system under development.

The protagonist model is derived from the system description and from scenarios worked out. The 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 this, the PAN is used [19]. A specification in the PAN is initially described through a directed graph (GPM) where the nodes stand for protagonists and the edges connecting nodes identifying the relationships between the protagonists. Note that to carry out tasks, interactions between these protagonists must take place. As well, a protagonist may carry out a task not related to another one. Henceforth, there is a need to capture not only user actions but also system-generated actions. Thus using a notation that captures only the 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. The use of the PAN provides the designer with a high abstraction level notation that allows him/her to document the user interface design requirements.

It is advocated the need to start protagonist-oriented analysis and design at a high abstraction level where only the user intentions are captured. This makes the user interface design easier for handling and upgrading as the refinement process proceeds until detailed requirements are obtained and implementation decisions are made. 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. a trashcan icon), the 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.

Note that abstraction is a key principle in any discipline and software engineering is not an exception. Such a concept together with modularity [13, 14] are used as design guidelines in the protagonist-oriented approach.

3.2 Task Hierarchical Model (THM)

From the identification of the protagonists, their goals and their interactions at a high abstraction level, the GPM is obtained. Such a model is derived smoothly based on the interaction scenarios where each protagonist has its major goal identified and is capable of performing some particular tasks.

With the GPM at hand, a task set for each protagonist is captured via the THM. Task hierarchical model shows how tasks are hierarchically organized. A task can be decomposed into sub-tasks as illustrated in Figure 3a. Additionally, each task can consist of one or more actions. It may also contain procedures with single or multiple actions. Actions, in turn, might be performed on objects as shown in Figure 3b. The THM is aimed at describing the hierarchy of tasks for each protagonist. In addition, each protagonist has a major goal associated with itself. The goal is viewed as a state that the protagonist aims to achieve.

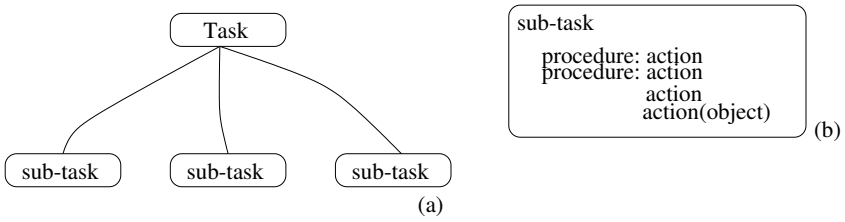

Figure 3: Decomposition of a task.

Consider the example of the game so-called *tic-tac-toe* where the major goal is to establish a sequence of three pieces. Tic-tac-toe is played with a square board containing a 3x3 array of positions (3 lines, 3 columns). A move consists of selecting a position and click on it to mark that position with, e.g., an X or O. To win a game, a player needs to establish a sequence of three of its own marks (X or O) horizontally, vertically or diagonally.

Tic-tac-toe is a turn-based game because a player is only allowed to make a move when it is in his/her turn. Tic-tac-toe requires two players for a game. Note that tic-tac-toe is a networked Web game, i.e. a multiplayer game which involves two players that are able to play the game together and interact with each other via their Web connection. Figure 4 illustrates a scenario where the player X makes a move and the player O updates the game state. Tic-tac-toe starts out by setting up an empty game board and giving one of the players the first turn. Once one of the players is enabled (e.g. the player X), he/she makes his/her move and the play shifts back to the other player. This exchange of turns continues until a win, loss, or tie takes place. When the game ends, you can start a new one if there is an opponent or you could play against the computer if an artificial intelligence (AI) strategy is used. In the latter case a goal is mapped onto tasks of running the game, waiting your turn, finishing the game, and so on, as shown in Figure 5. Within each THM, different elements are represented as follows:

- *Protagonist goal* - viewed as a state that the protagonist aims to achieve. For example, the protagonist goal of *play* shown in Figure 5.
- *Protagonist task* - refers to a defined piece of work, usually of short or limited duration, assigned to or expected of a protagonist. An example of a task is *update the game state*.

- *Procedure* - statement that provides some functionality for the system. It consists of single or multiple actions. In the task *update the game state*, a statement that may be used is *get the game status*.
- *Action* - mechanism used to control tasks. It is part of a procedure. The *notification that a player move was made* is an example of an action.
- *Object* - entity defined by its properties and is associated with one or more actions. The *status* of the game is an object.

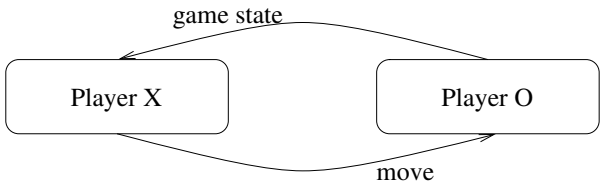

Figure 4: GPM for tic-tac-toe.

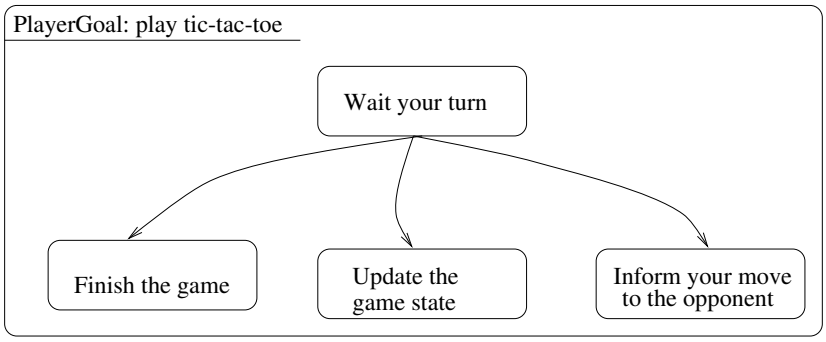

Figure 5: THM of tic-tac-toe.

3.3 Task Coordination Model (TCM)

TCM is part of the PAN and is used to describe how protagonist tasks are related to one another. Each protagonist has its tasks described through a TCM. TCM is a directed graph where nodes stand for tasks and edges connecting nodes represent the stimuli that trigger navigation among tasks. The idea is not only to show the tasks of a protagonist but also to reveal the relationship among them and to identify its major goal(s), i.e. how the task coordination for each protagonist takes place.

Moreover, the designer does not conceive this model at once. Initially, he/she identifies the protagonist goal(s) and tasks, then how they are related to each other and, finally, what stimuli cause navigation between tasks. In particular, the termination of one task can motivate a navigation to another task. Also, actions of other protagonists may cause the navigation between tasks in the TCM. In addition, the directed edges in TCM are labeled with stimuli caused by the actions performed by protagonists. Protagonist can exchange action(s) between each other. Additionally, protagonist actions can be performed on an object. Figure 6 illustrates the TCM for the player. Note that the TCM not only shows the tasks of the player but also reveals the relationship among them, i.e. how the coordination takes place.

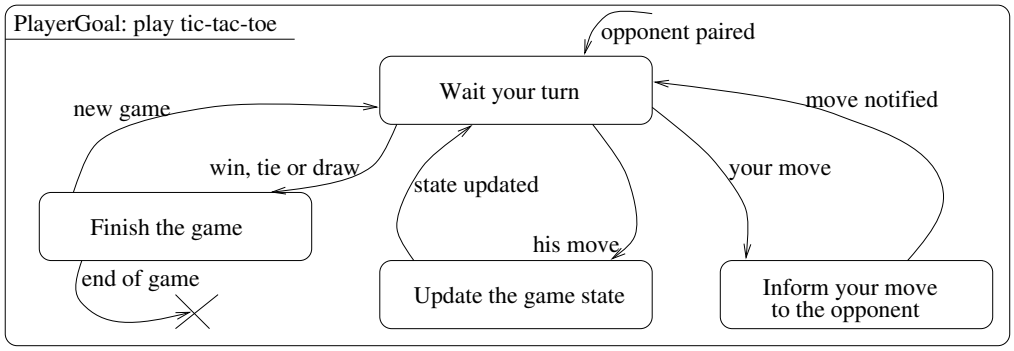

Figure 6: TCM for player.

3.4 Presentation Component Model (PCM)

The PCM defines how the TCM nodes, i.e. protagonist tasks, of the task coordination model are presented to the users. It may consist of a collection of UI objects, also called widgets, or windows. As well, appearance and behavior of the window should be consistent. On the one hand, the primary window is the one through which the other windows in an application are derived. On the other hand, secondary windows are those generated through a primary window. Windows can be used for different independent tasks and are an excellent way to allow users to work on more than one part of an application at once, or even to work within more than one application at a time. For example, a collection of UI objects can be laid out on a window. A UI object can be either a primitive UI object like text or button or a composition of objects. A kind of composite UI object is a presentation entity/object that depends on the task of the associated TCM. The most frequently used UI objects are: text, image, audio, button, video, anchor, and anchored collection. In addition, the presentation object is a UI object that depends on the task of a TCM and is the composition of other UI objects. The UI objects (or widgets) are the interface elements that allow to present all the information handled by the application component and receive the user input through a keyboard or mouse. These UI objects are usually provided by toolkits.

Within this context, the callback mechanism can be used by toolkits to carry out the communication between the widget collection software and the other interactive system components. A widget invokes a callback whenever the condition associated to the widget is satisfied. For example, when the instance of a widget(e.g. a button) is created, a callback is registered for that instance. Henceforth, whenever the user clicks on that button(condition), a callback is invoked.

Consider the example of the tic-tac-toe posed earlier. In that example, there are four tasks, i.e. wait your turn, inform your move, update the game state and finish the game. For each task, a screen shot is shown aiming to highlight the presentation aspects. The PCM for the player protagonist would show different views of the same window, that is the board of the game. The major goal of the PCM is to capture the kind of interaction style(s) associated with the TCM and provide information about the presentation aspects for the interactive system being developed. Such an information can be in terms of the PCM which highlights the presentation objects associated with the TCM. It can also suggest the use of radio-button, check-button, pop-up, and pull-down menus, forms, boxes or any other graphical interface types such as animation, video, audio and so on.

4 Case Study: Web Collaborative System

This section presents an example of a Web collaborative system, called *DirectionLeader* - a multiuser Web application - which involves multiple users at different locations. We start providing background issues on multiuser applications 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 Multiuser Applications

Multiuser applications that can be run on multiple machines are also known as *Network application*, which means that the applications are capable of enabling multiple users to play interactively on top of a network. In the case of networked Web applications, the communication between users is mediated by the Internet. In other words, in a networked Web application which involves two or more users, users are able to play the application together and interact with each other via their Web connection in a concurrent manner. In multiuser applications, the communication design can be affected by the way the collaborative system progresses, which is determined by the type of the particular application. Most applications fall into one of two categories:

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

DirectionLeader is an event-based application because users can make a move at any time as presented next.

4.2 Designing DirectionLeader

DirectionLeader is a multiuser Web application where the major goal is to lead the direction of a set of followers that can go to any direction under the leader command. For this application, we will consider each user as a player.

4.2.1 Describing the System

The system description is as follows. 7 illustrates the screen of our game.

> 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 and their move directions are guided by a leader which identifies each participating player. That is, a leader is protagonized by each player that can change the follower move direction at his/her will. Thus, each leader can recruit their followers by making a selection out of 64 available followers on the system as well as can choose their move direction. 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 7). 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. The DirectionLeader is a event-based application because every player can make a move at any time. DirectionLeader can be played with any number of players. Each player can be run on any machine, that can be either the server's or client's machine. Every follower has a screen location and a direction. When you click near a follower, it sprouts a little rod and prepares to move in the direction from where you clicked. The speed at which your followers move is 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 also seen by any 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 your player name. While players keep making their moves, a server listens to changes in the direction attribute and when there is a direction available, it moves the location according to the direction of each tick.

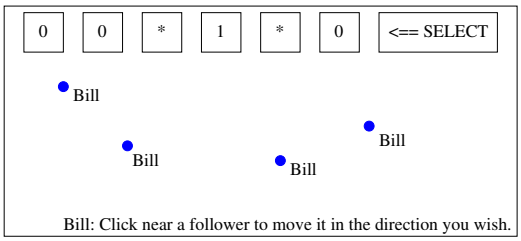

Figure 7: Board of DirectionLeader.

Once the system description is obtained, we identify system protagonists as follows. We also present the system components and the architectural model of DirectionLeader.

4.2.2 Identifying the System Protagonists

Protagonists are all the components that play roles in an interaction scenario. 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.

Considering our case study, each player interacts with the corresponding player interface component and notifies his move to all other players involved in the Web collaborative system. A player is protagonized by a user generally at a different location. When a move is made, it is sent out to the coordinator that works in background listening to the moves and propagating them to all the other players. The propagation only takes place when the moves occur which is a nicer solution differently of a polling approach that causes lots of unnecessary network traffic. Figure 8 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 the moves of all the players 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 is that where everyone knows each other. It 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, then, requests the service(s) through these interfaces. This kind of collaborative relationship is the one which requires more from a designer. 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.

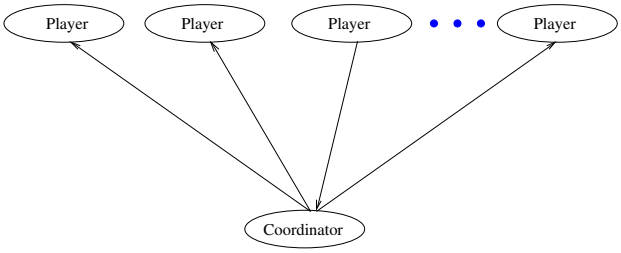

Figure 8: DirectionLeader protagonists.

Figure 8 shows the protagonists identified for DirectionLeader. To do that, we use PAN (Protagonist Action Notation) to provide the designer with a high abstraction level notation that allows him/her to document the UI requirements.

4.2.3 Developing 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 the architectural model of the interactive 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 9. Therein, one of the 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.

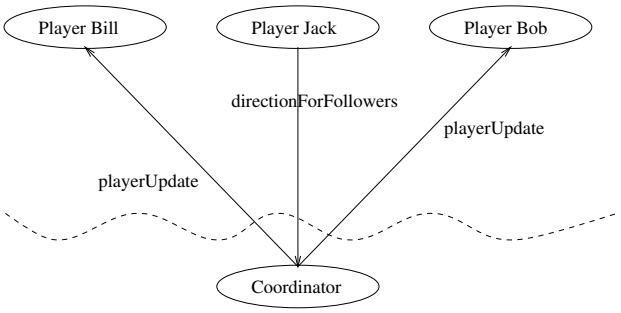

Figure 9: Scenario with application protagonists.

4.2.4 Identifying the Architectural Model

At this step, we describe the system in terms of their protagonists. Figure 9 gives 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 serving or clienting at any time. Nevertheless, note that all of these peer components always see an up-to-date view of the shared workspace. To do so, a form of *caching* is used to guarantee that synchronized full replicas of the peers are maintained on each client. So we identify the *peer-to-peer* architetural model for this system.

4.2.5 Describing DirectionLeader Tasks

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.

Figure 10a illustrates the Coordinator tasks through a TCM. It 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. Figure 10a represents the tasks and their relationships of only one protagonist, i.e, it describes tasks of and how they are coordinated by the coordinator protagonist.

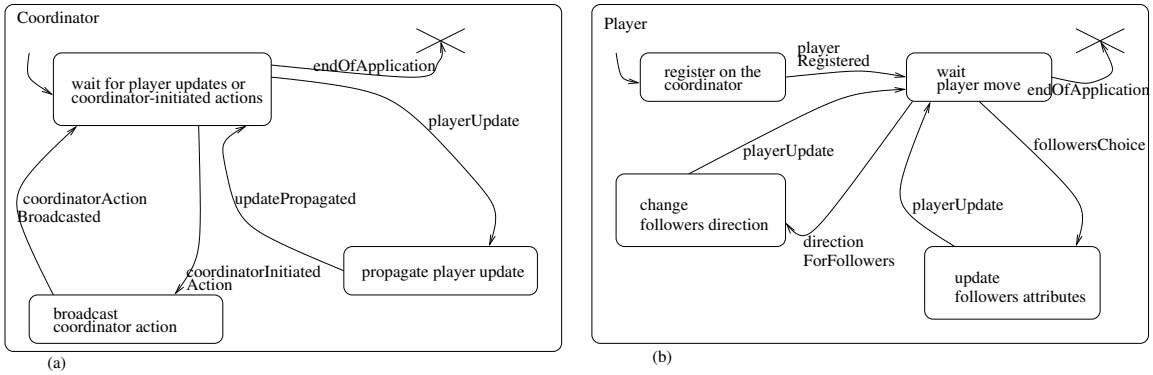

Figure 10: TCM for coordinator and Player.

Note that the designer does not conceive this model on the fly. Initially, he identifies the tasks, then how the tasks are related to each other, and finally what stimuli cause navigation between tasks. The users of the system are also involved within this process as reported in [19]. Furthermore, Figure 10 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 system status.

Directed edges in TCM are labelled with stimuli generated by protagonists. A stimulus can be passed from one protagonist to another. Additionally, stimuli can carry data between protagonists. The reader should note that a coordinator does not interfere in an application. It is in charge of mainly detecting player updates and propagating them to the other players.

The other DirectionLeader protagonist is a player. A player represents a user interface component which receives actions from its associated user and maps them into moves. Figure 10b gives the TCM for a player. Note that the main task of the players is to inform the coordinator about their moves. Each move can either be a follower number selection or a follower direction change. Moreover, a player needs to reason about the application in order to carry out a new move.

5 Related Work

Several analysis and design approaches that make use of models have been reported in the literature. Model-based design has been used to create interface designs represented at a high abstraction level. Design is described through models by using appropriate notations which capture aspects related to human-computer interactions. This approach provides a high abstraction level to represent the design as advocated by Holtzblatt and Beyer [7, 8].

Within this context, Foley et al [5] describe a User Interface Design Enviroment (UIDE) that offers facilities for designing the interface, not necessarily for managing the interaction at runtime. The UIDE comprises *(1) an application model* which defines the allowed user actions and their effects on application objects, *(2) an interface model* that describes user actions in terms of objects, *(3) pre- and post-conditions* associated with actions, and *(4) objects* that are defined in terms of their attributes. Nevertheless, the language-based nature of many UIDEs is somewhat oriented towards programmers rather than human factors experts and the environments do not directly support other aspects of the design process such as task analysis or scenario modeling.

The Method for USability Engineering (MUSE) is another approach described by Lim et al [10]. MUSE integrates human factors techniques with the Jackson System Development (JSD) method. MUSE uses the JSD notation to describe task hierarchies. It is a complex method, involving many stages, with relations to JSD defined at several points. In addition, no tool support is provided and the modeling notations are not specifically designed for their purpose.

Another approach is TRIDENT [21] which is an integrated methodology based on tasks. It is a modeling approach similar to task knowledge structure (TKS) [9]. TRIDENT allows automatic generation of user interfaces from abstract models. However, it places greater emphasis on tool support for the design.

It is worth observing that all these approaches require a greater number of modeling stages. In addition, note that a model-based approach like ours aims at improving the support of both the development life cycle and the design analysis. Moreover, an improvement in terms of usability and design traceability is achieved since the models of our approach evolve from user requirements to user interface, that is, a conceptual design. In other words, our protagonist-oriented approach involves explicitly defining the ideas or concepts underlying the user interface or a product.

6 Concluding Remarks

This paper presents a protagonist-oriented approach for capturing user interface design. Target applications are those in the Web collaborative systems context. This emerging class of application has been a result of the Internet era where a diversity of users and

interaction styles has required more and more from designers. One of our major concerns was to find out a way of identifying the social scenario where the computer technology can provide support. A case study involving a Web collaborative system has been presented to illustrate the application of our protagonist-oriented design approach. Other applications we have worked out has been reported in [19, 20, 16, 17]. The work presented in this paper is part of a major project which aims at both mapping a user interface design into an interface software design [16, 18] and providing an interactive systems development methodology. Therein, we address the derivation of interface software design from the UI design.

7 ACKNOWLEDGEMENTS

The authors would like to thank the financial support from Brazilian Council of Research (CNPq).

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] J. Blomberg, J. Giacomi, A. Mosher, and P. Swenton-Hall. Ethnographic Field Methods and Their Relation to Design. *D. Schuler and A. Namioka (Eds.): Participatory Design: Principles and Practices, Lawrence Erlbaum Associates, Publishers*, pages 123–155, 1993.

[3] A. Clement and P. Van den Besselaar. A Retrospective Look at the PD Projects. *Communications of the ACM*, 36(4):29–37, June 1993.

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

[5] J. Foley, W. Kim, S. Kovacevic, and K. Murray. UIDE - An Intelligent User Interface Design Environment. In J. Sullivan and S. Tyler, editors, *Architectures for Intelligent User Interfaces: Elements and Prot otypes.* Addison-Wesley, 1991.

[6] J. Greenbaum. PD: A Personal Statement. *Communications of the ACM*, 36(4), June 1993.

[7] K. Holtzblatt and H. Beyer. Contextual Design: Principles and Practice. *D. Wixon and J. Ramey (Eds.): Fields Methods Casebook for Software Design, Wiley Computer Publishing*, pages 301–333, 1996.

[8] K. Holtzblatt and H. Beyer. Contextual Design: Defining Customer Centered Systems. *Morgan Kaufman Publishers*, 1998.

[9] P. Johnson. *Human Computer Interaction: Psychology, Task Analysis and Software E ngineering.* McGraw-Hill, 1992.

[10] K. Y. Lim, J. B. Long, and N. Silcock. Integrating Human Factors with the Jackson System Development Metho d: An illustrated Overview. *Ergonomics*, 35(12):1135–1161, 1992.

[11] Comm. of the ACM. Special Issue on Health Care Information Systems. *Communications of the ACM*, 40(8):80–117, Aug 1997.

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

[13] D. Parnas. On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12):1053–1058, Dec 1972.

[14] D. Parnas, P. C. Clements, and D. M. Weiss. The Modular Structure of Complex Systems. *IEEE Transactions on Software Engineering*, 11(3):259–266, Mar 1985.

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

[16] A. M. Silva Filho, R. S. M. Barros, and H. K. E. Liesenberg. Designing User Interface for Web Interactive Systems. *Proceedings of the 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET 2000), Richardson, Texas, USA*, pages 7–14, Mar 2000.

[17] A. M. Silva Filho, R. S. M. Barros, and H. K. E. Liesenberg. Designing Multiple User Interfaces for Internet-based Interactive Systems. *Proceedings of the French/British Conference on Human Computer Interaction, Lille, France*, pages 17–14, Sept 2001.

[18] A. M. Silva Filho, R. S. M. Barros, and H. K. E. Liesenberg. From HCI Design to User Interface Software Design: Towards Automatic Derivation. *(submitted)*, 2002.

[19] 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, Brasil*, 1:276–296, Aug 1998.

[20] A. M. Silva Filho and H. K. E. Liesenberg. Designing Synchronous User Interface for Collaborative Applications. *Interactive Systems Design with Object Models, N. Nunes et al (Editors), Springer-Verlag LNCS, 1743*, pages 267–287, 1999.

[21] J. Vanderdonckt and F. Bodart. Encapsulating Knowledge for Intelligent Automatic Interaction Objet cts Selection. *Proc. Confererence on Human Factors in Computing Systems: InterCHI '93*, pages 424–429, 1993.

[22] 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.