A Framework for Awareness Support in Groupware Systems

Manuele Kirsch-Pinheiro[a*], José Valdeni de Lima[b], Marcos R. S. Borges[c]

[a] *Laboratoire LSR - IMAG, France*
[b] *Instituto de Informática - Universidade Federal do Rio Grande do Sul, Brazil*
[c] *NCE & IM - Universidade Federal do Rio de Janeiro, Brazil*

Abstract

This paper introduces a framework for awareness support in groupware systems. Awareness gathers the knowledge on group, its activities and its overall status. Awareness support is an important feature for groupware systems. It provides a context for individuals contributions, improving those contributions and avoiding contradictory interactions among group's members. Despite its importance, awareness support is not systematic and developers must build it from scratch for each new application. The framework presented here addresses this issue. This framework, called Big Watch (BW), intends to support groupware implementers to easily past event awareness. It has been designed to develop new groupware applications and to improve existing ones. This paper presents the features and the structure of BW, and describes two applications that use it.

Keywords: awareness support, groupware systems, groupware design, framework, cooperative work and collaborative design.

1. Introduction

To get most things done in an organization today requires a great collaborative effort. Knowledge sharing is the key factor of collaborative environments, especially those which

[*] Corresponding author: BP 72 – LSR – IMAG, 38402 Saint Martin D'Hères CEDEX, France. Phone: +33476827274. Fax: +33476827287. Email: Manuele.Kirsch-Pinheiro@imag.fr Author supported by a grant from CAPES/Brazil.

support design tasks, due to its knowledge exchange nature. To be shared, knowledge has to be externalized and made visible to potential recipients, either humans or software agents.

There are many articles dedicated to knowledge representation and ways to make it explicit in design environments (Nunamaker et al., 2001; O´Leary, 1998). A member of a design group makes his knowledge explicit when he/she traduces it into some representation supported by the environment. However, it is not enough to make knowledge explicit; it is also necessary to provide group members with mechanisms that inform them the knowledge is there. Only when a group member perceives the new knowledge the socialization process may occur (Nonaka and Takeushi , 1995).

The contextual information about group members' work is usually provided by awareness mechanisms (Dourish and Bellotti, 1992). According Paul Dourish (1992), '*Awareness is an understanding of the activities of others, which provides a context for your own activity. This context is used to ensure that individual contributions are relevant to the group's activity as a whole, and to evaluate individual actions with respect to group goals and progress'*. Awareness mechanisms are therefore essential to group support systems in order to transform irregular interactions of group members into a consistent and perceptive performance over time (Preguiça et al.,2000).

On the other hand, depending on its quantity, awareness information can also be very distracting and harmful to individual activities when the mechanism does not accommodate individual needs and preferences. Information overload is a well-known phenomenon resulting from this inadequacy. Thus, awareness support should be a major concern when designing groupware systems.

However, systematic solutions for the awareness support are not common. Due to their complexity, awareness mechanisms are either inexistent or provide limited support to knowledge perception. Besides, the implementation of an awareness mechanism requires a great effort from groupware developers when they build this support from scratch. Groupware implementers often do not have any framework or toolkit to help them, and the reuse of code from other applications is not an easy task.

We designed a framework, called BW, to help groupware developers to implement awareness mechanisms in groupware systems. In its first version, it aims to supporting past events awareness. It has been designed to be flexible enough to improve existing groupware applications and also to help implementers to incorporate awareness mechanisms when building new applications. In this paper, we describe the characteristics and the structure of the BW framework.

Our goal with BW is two fold. First, we want to facilitate the work of groupware implementers by providing them with a framework that incorporates most of the awareness functionality. This should reduce the work required to implement a complex mechanism and save time to dedicate other important issues in the application. Second, we want to develop a framework that can be extended to incorporate new mechanism and filters, whenever the groupware application requires them.

The paper is organized as follows: first, we discuss the importance of the awareness support in groupware systems and present some related works. Then, the framework is presented, first through its main characteristics, then through the presentation of its structure. Once the framework is presented, we show how to use this framework and describe some applications that already use it. Finally, some conclusions are presented.

2. Awareness in Groupware Systems

The work executed within a group produces better results when there is a harmonic interaction among group members. This harmony depends on the level of understanding among these members. To reach this understanding, the group needs four types of support: (1) communication among the participants; (2) coordination of their activities; (3) a "group memory", which records the group's common knowledge, such as the interaction between the participants and the products developed by them; and (4) awareness support (Dias and Borges, 1999).

Providing computer-based mechanisms for supporting awareness has been shown to be of a vital importance in the design of collaboration support systems. Being aware of the colleagues and their activities is very important to make the work more natural and fluid (Gutwin and Greenberg, 1998, 1999). Moreover, keeping users informed about what is happening to the resulting product will reduce the risk for double-work and integration problems (Farschian, 2001). Besides, making awareness information available may increase the group's shared knowledge.

Thus, the need of awareness support in cooperative environments is obvious, and this support should be one of the main concerns when designing a groupware system. Moreover, the communication among group's members can also be intermediated by this awareness support. Actually, awareness is a design concept that holds promise for significantly improving the usability of the groupware systems. Nevertheless, with a few exceptions, this support usually involves particular solutions to specific domain problems and isolated approaches that are difficult to generalize to other situations. As a consequence, designers must re-invent awareness, for each new application, based on their experience of what it is,

how it works, and how it is used in the task (Gutwin and Greenberg, 2002). A similar situation occurs in groupware maintenance when the groupware designers need to improve the awareness support of their system. In both cases, new applications and existing ones, awareness mechanisms are built from scratch.

To help groupware designers in this task, Kirsch-Pinheiro et al. (2001) identified some important characteristics needed to provide such support. These characteristics are organized into 6 questions (what, when, where, who, how, how much), each one identifying crucial aspects of awareness support in cooperative systems: what information to present, when this information is produced/presented, where it is produced/presented and how, who is working and how much information about all this should be presented to the user (see Figure 1). These aspects are analyzed within two environments: synchronous and asynchronous. This division is important because systems in synchronous environments have different needs than asynchronous ones. For example, people working simultaneously in a shared workspace need to know what their colleagues are doing at this precise moment, involving information like their mouse movements and workspace position, while people working in an extended period of time do not need such precision.

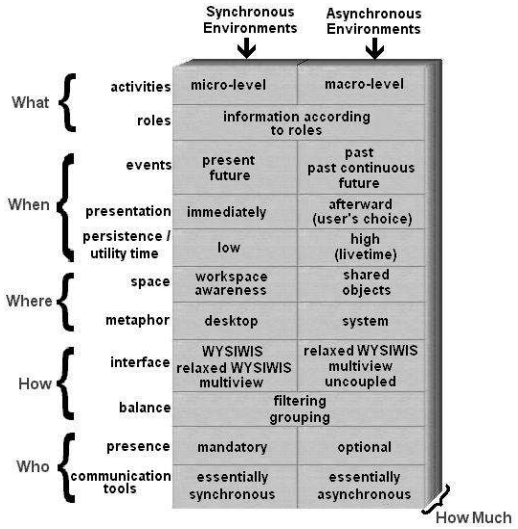

Figure 1: Important characteristics for awareness support

However, among all the questions presented above, the «when» question is the most relevant to this work. The “when” question tells when the cooperative activities executed by the group are produced and when the awareness information produced by those activities are presented to the end user. Depending on when each one of those activities happens, it may be more or less important to the group. For example, when working asynchronously or through many sections, a group will need to be aware of its past activities to keep in mind the evolution of the work to reach the goal. That is what we call "past event awareness": the awareness information about the activities performed in the past, whose results may have changed or not be valid anymore.

The support to past event awareness is a very important feature in groupware systems, especially in those systems that deal with asynchronous interactions or multi-section works. It is important mainly because the overall information about the evolution of the collaborative activities, such as the evolution of shared data and the users' actions, may improve each user's contribution (Preguiça et al.,2000).

3. Related Work

Although important, just a few groupware applications have past event awareness support. An example is the POLITeam Project (Sohlenkamp et al., 2000). This project intents to develop a groupware system supporting distributed and asynchronous cooperative work. It includes some features for past events awareness support, such as an event history dialog window. However, the solutions proposed by POLITeam are very specialized to its application (the Germany government), and they cannot be easily adapted to other situations.

In addition, there are not many tools for helping groupware developers to build awareness support. Among them, we can name the COPSE infrastructure, which also includes a framework, for cooperative software design. It allows the development of new groupware applications that can be integrated through the infrastructure (Dias and Borges, 1999). Although COPSE provides a group memory structure and some awareness support, there is no specific support for past event awareness.

Similar to COPSE, there is Habanero, a collaborative framework and environment that allows users to interact through a variety of applications that share state and events. The Habanero framework is an API that allows developers to create collaborative Java applications. It provides methods to build or convert existing applications into collaborative applications (NSCA Habanero, 2002). However, Habanero has a rudimentary support to past event awareness, limited mainly to session record/replay capabilities.

Moreover, there are in the literature toolkits for the development of synchronous applications. However, in their majority, they do not provide past awareness support. An example is GroupKit (Roseman and Greenberg, 1996, 1997), a toolkit for building

applications for real-time, distributed computer-based conferencing. It includes many widgets for workspace awareness support, but no past event option is proposed .

The framework presented here does not supply a structure for complete groupware applications. It is dedicated to past event awareness support. For example, it cannot inform that an activity will be finished soon (future awareness). Actually, the framework BW was designed to supply a systematic support to past event awareness. The next section presents this framework with its main characteristics and structure.

4. The Framework BW

The framework BW has been designed to provide a flexible mechanism to support past event awareness. This flexibility is its primary concern. Indeed, the framework BW was designed to supply past awareness support for existent groupware systems that need this support and also to build new groupware applications with this support.

In the next sub-sections, we show how the framework BW reaches this flexibility, using an event-based mechanism and a layered structure.

4.1. Event-based awareness

Any groupware application has its own set of activities that should be accomplished by the group. The awareness support should be adapted to those activities and to the group's goals and structure. Thus, the awareness support is tightly coupled to the groupware and its characteristics, such as members' experience, atomicity of activities, work process (workflow), etc.

To remain close to the groupware needs, and yet keeping its flexibility, the framework BW adopts an event-based awareness mechanism based on a three-layer structure. It is called an event-based awareness mechanism, because all awareness information is based on

events. Those events represent the group's activities, which were executed within the groupware application.

In the framework BW, this event-based awareness is organized in three phases (Figure 2): registering, monitoring and notifying. In the first phase, the groupware registers in the framework what events are interesting for awareness purposes. It is done by passing to the framework BW a sample instance of each expected event, an object to be used as an example by the framework BW. This example object is used to identify similar objects during the next phase. In the second phase, monitoring, the activities take place in the groupware, and once one of these activities is executed, the groupware can pass to the framework BW the event related to this activity.

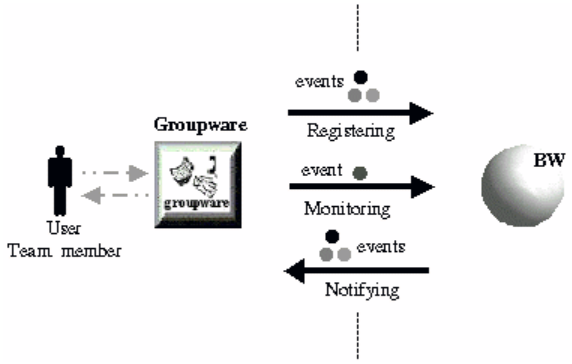

Figure 2. Registering - monitoring - notifying cycle

At this point, we can see that the framework BW works as an external element to the groupware. In fact, it acts like an encapsulated component: the groupware can safely use it without knowing exactly how it works internally. The framework BW interacts with this application only through the events, which are registered by the application and recognizable by the framework. Thus, the framework BW can be kept independent of the groupware application, keeping also its flexibility. As a consequence of this event-based

awareness, the groupware designer must identify what activities are important for the awareness support and define the corresponding events.

For example, considering a cooperative editor that imposes to the author a specific order for the text (abstract, introduction, chapters, conclusion and bibliography), the developer of such editor could register in the BW framework a prototype instance of an event «Conclusion done» to indicate that the conclusion part of the text has been done. Then, every time a group member finishes the text conclusion, the editor can produce and pass to the BW framework the event that describes that specific activity.

The last phase presented in Figure 2 is to notify the user (team member) about the awareness information. It consists in informing the user what has occurred in the group work. However, it can be easily seen that presenting the information about all activities may produce a large set, which is difficult to deal with. Such high amount of information may not be desirable. The user shall not spend much more time with awareness information than working. Besides, the user may not be interested in awareness information about all group's activities, but only a subset of these activities. Thus, it is important to adapt the awareness information to the user's interest and to his/her role profile.

Indeed, roles and awareness information have a very close relationship. Users need awareness information to better play their roles, and all roles do not need the same information. The awareness information presented to a user should be adequate to the user's role and preferences. For example, a user playing an author role in a cooperative authoring environment needs information different from that required by the team coordinator in the same environment. A coordinator should be informed about the overall work progress. He/she needs a global view of the group activities, the defined tasks and deadlines, to take

his/her decisions and guide the team efforts (Kirsch-Pinheiro et al., 2001). If the coordinator has an adequate awareness mechanism, which presents relevant information, the coordination task will be easier (Borges and Pino, 1999).

In order to adapt the awareness information to the user and role's needs, and, at the same time, to avoid an information overload, the framework BW executes a filtering of the available awareness information, based on profiles. A profile specifies the user's or role's preferences. It describes which activities, among the group activities, should be notified, according to his/her preference. These profiles indicate which activities are significant and the time interval during which they are significant. The framework BW defines three types of profiles: those associated with a team member, a role, or both (a team member playing a role). Based on the selection, the framework filters the available information, in such a way that only events that are indicated by one of those profiles are presented to the user. Thus, by setting their profiles, team members indicate about which activities he/she wants to receive awareness information. A similar mechanism has been proposed by David & Borges (David and Borges, 2001). To illustrate this aspect in our example of a cooperative editor, a user who sets into his/her personal profile the event «Conclusion done», is notified of all «Conclusion done» events performed inside his/her group. A coordinator receives the same information whether his/her role profile indicates that all events related to 'terminated sections' should be informed.

4.2. Framework Description

The processing of the phases Registering, Monitoring and Notifying is done by a structure of three layers, namely storage, control and user interface. Each layer is in charge of handling several aspects of this processing: the storage layer handles the storage of the

awareness information. The user interface layer manages the presentation of the awareness information to the user. The control layer is the most important layer. It handles the groupware requests, like the events registering and monitoring, and it also controls the filtering process.

Those layers have been organized in four independent packages. Each package assumes the responsibilities of the corresponding layer, and communicates with other packages by well-defined protocols. These protocols are, in fact, facade classes, an application of the design pattern Facade (Gamma et al., 1994). These facade classes represent the package functions for outside classes. Thus, each package knows only the facade classes from the other package, not their internal structure, keeping them independent.

Besides the presence of facade classes, the packages have another aspect in common: they manipulate the same information. All packages handle information about events, roles, profiles and users. So, in order to keep their independence, the information has been isolated in a fourth package, called Kernel. The Kernel package is of crucial importance to the framework BW, because it describes the awareness information manipulated by this framework. The next sub-sections will summarize each one of those packages, showing the most important features of each one.

4.2.1. Kernel. The kernel package describes the information manipulated by the framework BW. It constitutes the data model of the framework. Since this model is used by all packages of the framework, the kernel package is the only open package. Its content is

known by the other packages, since they need this knowledge to carry out their tasks. Figure 3 shows the main structure of the framework BW.

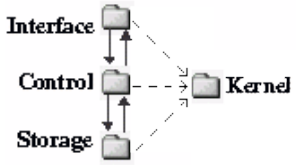

Figure 3. Framework BW structure

The most important information classes defined in the kernel package are events, users, roles, profiles and the group. All those classes are directly linked together, as shown in Figure 4. The group, represented by the class «Group», aggregates users (the team members), represented by the class «Members», and the roles (Role) that the users can play. This way, the fact that a user can play many roles in the same group can be modeled, as well as the dependency between roles and the group (the roles are defined for a group, which aggregates a set of possible roles).

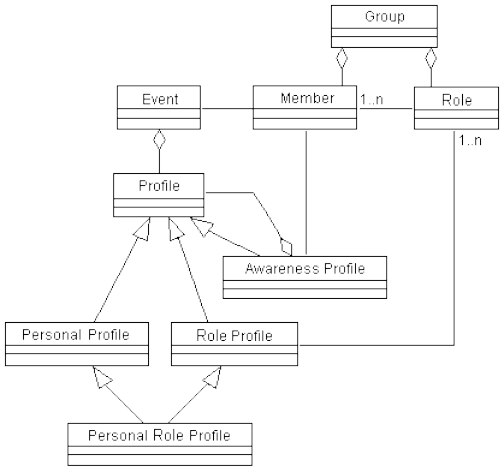

Figure 4. Some Kernel package internal classes

Besides this relationship between group, users and roles, there is also a relationship between user/roles and profiles. As we mentioned before, the framework BW defines three

kinds of profiles: the users' personal profiles (Personal Profile), the roles' profiles (Role Profile), and the combination of both, or the users' personal preferences when playing a role (Personal Role Profile). Each of these profiles is composed of a set of events, which represents the activities subscribed by the user or the role in that profile. All the user's applicable profiles are aggregated under the «Awareness Profile» class that composes the complete set of preferences applicable to the user.

Finally, there are the relationships between events, users and profiles. Events (Event) represent the activities performed by the users in the group. Consequently, there is a link between event and user objects. This link indicates which user was responsible for the event execution.

Besides the classes presented above, the kernel package also defines other classes: the register class, that keeps the set of registered events, a time interval class that is mainly used by events and profiles, and a super class that defines the framework BW basic element, which is specialized by the other classes in the kernel package.

4.2.2.Storage. The storage package is in charge of keeping the awareness information in a permanent base. The storage package is the interface between the framework BW and the database. It provides other packages with simple services, such as the saving and recovering of objects defined in the kernel package. By using these services, the other packages do not need to know anything about the database or the storage medium used. They only know the storage facade, keeping their independence from the storage device package.

To keep the flexibility of the framework BW, the storage package should not be linked to a specific DBMS. For this reason, the storage package uses the Bridge design pattern (Gamma et al., 1994) to implement the direct access to the database and media. This pattern separates the object abstraction from its implementation. Thus, we separate the abstraction of the database and media in a class (Strategist) from its implementation in a second class (Implementor). As a result, the framework BW can be easily adapted to the database and media used by the groupware application, by specializing «Implementor» class (Figure 5).

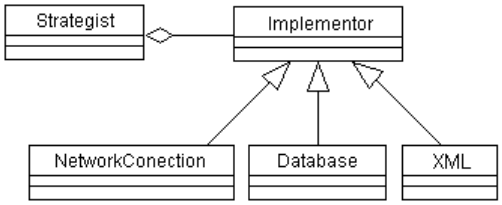

Figure 5: Storage package details

4.2.3.Control. The control package is the «central processing unit» of the framework BW. It receives and handles the groupware requests, and delivers the information and services to the other packages. For example, the control package receives from groupware the events produced by the group. It handles these events, verifies if they are registered events, and then sends them to the storage package to be saved.

The control package also manages the information filtering, needed for the notifying phase. It takes the user's applicable profiles (personal profile, personal role profile and role profile) and processes it in order to get a unique set of interesting events. This set is used to recover, through the storage package, the events produced by the group, which are passed to the user interface package, where those events will be presented to the user.

This filtering process is done by one class (Awareness) in the control package. It captures the profiles applicable to the user from his/her awareness profile and merges the set of events present in each profile, creating a final set of events. This final set of events is used to recover the available events from the Storage package. In addition, this class merges the time intervals associated with each user's profile. Each time interval is, indeed, an Allen's interval (Allen, 1983), so their merge returns a new interval, which is the union of each individual interval. Thus, the final set of events presented to the user reflects the user's preferences and role's needs. Furthermore, by isolating this processing in a single class, it is very easy to create new variations of this filtering process by specializing this class.

4.2.4.Interface. The last package, called Interface, handles the presentation of the awareness information to the user. The user interface is a delicate topic for a groupware. It is through its interface that the user interacts and cooperates with other team members, and it is also through the interface that users receive the awareness information. If the interface is notwell-suited, the user may not assimilate the awareness information.

Therefore, the user interface should be adapted to the groupware and to the information presented. It should also be integrated within the groupware interface. The user should not perceive the awareness support as a different system, but just perceive everything as the groupware.

The framework BW cannot define just one user interface encompassing all situations. Consequently, the interface package has been defined using mostly abstract classes. We defined one central class (Interface) that receives information to be presented from the control package, and two abstract classes, that represent the user interface elements. These

classes are organized in a container/contents structure: the container (class GUIElement) aggregates many contents objects (GUIEvent), each one handling the presentation of a specific set of events. Thus, the groupware designers can define a user interface adapted to the awareness information and groupware application, by specializing these container/contents classes, according to the events represented and the groupware interface. For example, he/she can define a hierarchy of colors for explicitly inform the importance if the events presented. As a result, groupware designers can better integrate its interface within the framework BW.

Moreover, it is also through the user interface that team members are able to manipulate their own profiles. Once again, groupware designers should define an interface adapted to the groupware and its activities.

5. Connecting the Framework BW

The first version of the framework BW was implemented in Java following the description presented in the previously sections. The official distribution[1] contains four Java packages, which correspond to the four packages presented above.

However, these packages do not specify how to integrate the framework BW with a groupware system. In fact, those packages present only a small set of "entry points", e.g. classes that must be specialized by the groupware developer when connecting the framework to the groupware. Those classes are: (1) the user interface classes, which should be adapted to the awareness information presented and to the groupware interface; (2) the storage implementation, which should adapt the framework to the real storage medium and database used by the groupware.

[1] available at http://www.inf.ufrgs.br/~manuele/BW/.

In order to effectively connect the framework BW to the groupware system, we introduced a new element, the mediator. This mediator centralizes every call coming from the groupware, acting as a general interface for the framework BW. This mediator is, actually, a combination of the design patterns Mediator and Singleton (Gamma et al., 1994), whose respective goals are to reduce the number of interconnections among the objects, and to propose an unique instance accessible for all "client instances" through a well-known interface.

This mediator, then, is organized in two levels. At the first level, is found the definition of general mediator's functions, formed by class methods. In the second level, lies the concrete mediator, which really implements those methods especially for the groupware. By using this structure, we unify all references to the framework into a unique class. As a result, we reduce the complexity of the references needed by the groupware, since such references are limited to class methods in the mediator class.

Thus, to use the framework BW, groupware developers are constrained to implement some classes. These classes include the user interface and storage implementation classes, and the concrete mediator, which should be designed for that specific groupware. Once these classes are written, the developers need only to introduce in the groupware the right calls to the mediator.

6. Case Studies

The framework BW has been already used in some applications. The first application that uses it is the framework COPSE and the class diagram editor CUTE. CUTE is a cooperative class diagram tool developed with COPSE (Dias and Borges, 1999). Both,

COPSE framework and CUTE, do not provide a support for past event awareness, despite their need for this support. Cooperative software engineering is typically a multi-section work (the developers group need many sections to accomplish the software design and development), and past event awareness support in such environments is very important in order to keep track of the evolution of the software. Thus, we introduced this support by using the framework BW.

To introduce the framework BW into COPSE/CUTE system, we followed the process described in the above section. First, we implemented the DBMS access, by specializing the storage implementation class. In the new class created, we use a JDBC PostgreSQL driver to access a centralized PostgreSQL database, applying the same centralized architecture used by COPSE/CUTE (one central server to which all clients are connected). Then we created the user interface for this new awareness feature, by specializing the container/contents classes defined in the Interface package. These new classes (Figure 6) have been designed to be perfectly adapted to CUTE interface. As a consequence, users perceive only the CUTE groupware interface, now with a new "past awareness" feature. They do not perceive this new feature as a different system element.

In addition, we also built the concrete mediator for the COPSE/CUTE system. By using this mediator, we minimize our interventions into COPSE framework. Besides, the mediator also dynamically binds the new classes developed for this system to the framework. As a result, the framework can use those classes without any internal modification on the framework.

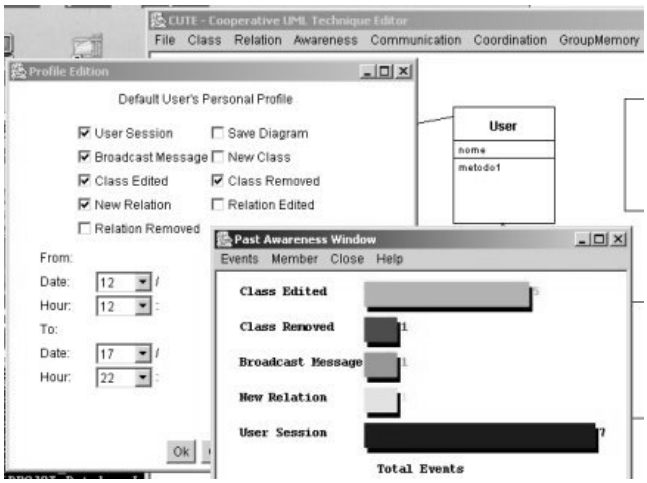

Figure 6. Screenshot from CUTE awareness support

During the use of the framework BW in the COPSE/CUTE system, we noticed two critical points. First, the design of the concrete mediator becomes very important, since it is the main connection between the system and the framework, and its complexity may increase according to the complexity of groupware application.

The second critical point identified is related to the registering phase, and more precisely, to the events definition. This definition necessitates to decide what activities should be objects of the past awareness support, and what information about these activities should compose the corresponding event objects. This process requires an analysis, by the groupware developer, of the real needs of the users. For the COPSE/CUTE system, we defined a small set of nine events, most of all are related to the edition of classes and relations in the UML class diagram edited by the group. Figure 6 shows the possible events on the profile window.

The framework BW has also been used by the CEMT Project (Kirsch-Pinheiro et al., 2002). In this project, the framework BW was used to build an awareness server. This server communicates with client applications (mainly Web editor applications, such as

Amaya (Vatton, 2001)), which send their awareness information through an XML based protocol, in a cooperative environment for e-learning authoring. The server keeps this information in a persistent database, which is also used by a workflow engine that guides the authoring process. When a client application requires the awareness information, it requests this information through the XML protocol. Once this information is received, the client application can present it to its users. Besides, the awareness information kept by the framework BW can also influence the workflow. In fact, by being aware of their colleagues' activities, users may anticipate other tasks, and dynamically change the workflow.

In CEMT Project, we also implemented the mediator class, represented by the awareness server itself. For the DBMS access, we reused the class implemented for COPSE/CUTE, by using the same JDBC PostgreSQL driver and database structure. However, for the user interface classes, we specializeed the Interface package. In the CEMT Project environment, the awareness server does not interact directly with the users, but with client applications through network connections. Thus, we adapted the Interface package to this distributed environment, where it interacts with many users connected through client applications.

The development of this awareness server in the CEMT Project shows a third critical point in the framework BW application: the developers' learning curve. In fact, groupware developers need some time to learn about the framework BW, its structure, and how use it. Only after this learning period, groupware developers feel comfortable with the design of the new classes allowing the use of the framework BW in their system.

7. Conclusions

One of the main goals of the framework BW was to provide awareness information with flexibility. This goal has been reached by the use of an event-based awareness mechanism, designed using an object-oriented approach. The framework is divided into four packages, which separate its services and its data model, keeping them independent. Besides, the use of an object oriented design allows the groupware designers to easily extend the framework BW through the specialization of the framework classes.

As a result, we reached a flexible and extensible framework, which has been used successfully in two different environments, the environment proposed by the CEMT Project, and the COPSE/CUTE application. These applications also showed us the overhead experienced by the groupware developers when using the framework BW. This overhead is related to (1) the conception of new classes that connect the framework to the application; (2) the amount of time that developers need to learn about the framework; and (3) the definition of the events, which is not an obvious task, since developers have to define the users' needs about past events awareness. If this definition fails, the system will not reach its goal of assisting users in their cooperative activities. However, the analysis of which activities are important for the awareness support is also required when developers are not using this framework. Hence, we may consider that the framework BW can help groupware implementers even in this point, since they will be able to dedicate more time to this delicate definition task, and not to design definitions encapsulated by the framework.

Thus, in theory, the framework BW can be integrated into any application that uses an OO approach for supplying users with past event awareness. The use of our framework reduces the development effort and provides awareness information in a unified way.

Therefore, awareness information composes the group's shared knowledge. The framework presented here allows a representation this knowledge and externalization of it. The framework BW constitutes, hence, a mechanism to make this knowledge available and to inform group's members about this availability.

The framework BW still has many possibilities to explore, due to its flexibility. It can be used, for example, to implement the idea of "awareness of awareness" (that is, supply information about who received an awareness information), or for "future awareness" (awareness information about the incoming events for the group). Implementations in other environments are under development.

References

Allen, J. F., 1983, Maintaining Knowledge about Temporal Intervals, Communications of the ACM, 26(11), 832-843.

Borges, M.R.S., and Pino, J.A., 1999, Awareness Mechanisms for Coordination in Asynchronous CSCW, Proceedings of 9th Workshop on Information Technologies and Systems (Charlotte).

David, J.M.N., and Borges, M.R.S., 2001, Improving the selectivity of awareness information in groupware applications, Proceedings of CSCWD'2001 (IEEE Computer Society), 41-46.

Dias, M.S., and Borges, M.R.S., 1999, Development of groupware systems with the COPSE infrastructure, Proceedings of International Workshop on Groupware (IEEE Computer Society, Cancun), 278-285.

Dourish, P., and Bellotti, V., 1992, Awareness and Coordination in Shared Workspaces, Proceedings of ACM Conference on Computer-Supported Cooperative Work (ACM

Press, Toronto), 107-114.

Farschian, B. A., 2001, Integrating geographically distributed development teams through increased product awareness, Information System Journal 26(3), 123-141.

Gamma, E., Helm, R., Vlissides, J., Johnson, R., 1994, Design patterns: elements of reusable object-oriented software (Addison-Wesley).

Gutwin, C. and Greenberg, S., 1998, Effects of awareness support on groupware usability, Proceedings of CHI'98 - Conference on Human Factors in Computing Systems (ACM Press, New York), 511-518.

Gutwin, C., and Greenberg, S., 1999,A framework of awareness for small groups in shared-workspace groupware, Technical Report 99-1 (Department of Computer Science, University of Saskatchewan, Saskatchewan). Available at: http://www.cpcs.ucalgary.ca/papers/1999/99-AwarenessTheory/html/theory-tr991.html. Access: September, 1999.

Gutwin, C., and Greenberg, S., 2002, A descriptive framework of workspace awareness for real-time groupware, Computer Supported Cooperative Work (Kluwer Academic Press). Available at: http://www.cpsc.ucalgary.ca/grouplab/papers/ Access: April, 2002.

Kirsch-Pinheiro, M., Lima, J.V., and Borges, M.R.S., 2001, Awareness em sistemas de groupware", 4 Jornadas Iberoamericanas de Ingenieria de Requisitos y Ambientes de Software (Centre de Información Tecnológica, San Domingos), 323-335.

Kirsch-Pinheiro, M., Telecken, T., Zeve, C., Lima, J.V., and Edelweiss, N., 2002, A cooperative Environment for e-learning authoring, Documents numériques 5(3-4), 89-114.

NCSA Habanero. Available at: http://www.isrl.uiuc.edu/isaac/Habanero/. Access: February, 2002.

Nonaka, I., Takeuchi, H., 1995. The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation (Oxford University Press, Oxford).

Nunamaker Jr., J.F., Romano Jr., N.C., Briggs, R.O., 2001, A Framework for Collaboration and Knowledge Management. In: Proceedings of 34th Hawaii International Conference on System Sciences - HICSS'01 (Hawaii, EUA).

O'Leary, D., 1998, Enterprise Knowledge Management, IEEE Computer 31(3), 54-61.

Preguiça, N., Martins, J. L., Domingues, H., and Duarte, S., 2000, Data management support for asynchronous groupware, Proceedings of ACM Conference on Computer-Supported Cooperative Work (ACM Press, Philadelphia), 69-78.

Roseman, M., and Greenberg, S., 1996, Building real time groupware with GroupKit, a groupware toolkit, ACM Transactions on Computer Human Interactions 1(3), 66-106. Available at: http://www.cpsc.ucalgary.ca/grouplab/papers. Access: January, 2000.

Roseman, M., and Greenberg, S., 1997, Building groupware with GroupKit, in: Harrison, M., eds., Tcl/Tk Tool (O'Reilly Press), 535-564. Available at: http://www.cpsc.ucalgary.ca/grouplab/papers. Access: January, 2000.

Sohlenkamp, M., Mambrey, P., Prinz, W., Fuchs, L., Syri, A., Pankoke-Babatz, U., Klöckner, K., and Kolvenbach, S., 2000, Supporting the distributed German government with POLITeam, Multimedia Tools and Applications 12(2), 39-58.

Vatton, I., 2001, Welcome to Amaya. Available in http://www.w3.org/Amaya/. Access: Aug, 2001.