AN ARCHITECTURAL PATTERN FOR COMPUTER-MEDIATED COMMUNICATION IN GROUPWARE SYSTEMS

Luis A. Guerrero, Sergio Ochoa
Computer Science Department, Universidad de Chile.
{luguerre, sochoa}@dcc.uchile.cl

Oriel Herrera
Informatics School, Universidad Católica de Temuco.
oherrera@uct.cl

David A. Fuller
Computer Science Department, Pontificia Universidad Católica de Chile.
dfuller@ing.puc.cl

Abstract: In the creation of groupware systems, a good design of communication mechanisms, required by the group in order to carry out its work, is essential, since there are many other design aspects that depend on it. These aspects have a major impact on the success or failure of the collaborative application. This paper presents a computer-mediated communication taxonomy for groupware systems, as well as an architectural pattern to support the design and construction of the communication mechanisms required by the groupware systems, bearing in mind the other application's design aspects.

1 INTRODUCTION

The communication design of groupware systems is very important, since there are many design aspects that depend on it [3]. These are, for example, awareness, session and users administration, floor control, notifications, among others. These aspects have a major impact on the success or failure of the collaborative application. This article describes and classifies some ordinary communication scenarios, which can be found in groupware systems. Also, an architectural pattern to support the design and construction of the communication mechanisms required by the groupware systems is presented.

In the communication design of groupware system it is important to distinguish between user communicational processes and system communicational processes. A *User Communicational Process* (UCP) is the process that sends user messages, or that which receives messages containing information to be interpreted by a user. A *System Communicational Process* (SCP) is the process that sends system messages, or that which receives messages with contents not be interpreted by a user, but to be processed by a system module.

All messages will be sent through a channel and they should have at least three components: sender ID, recipient ID, and message contents. Sender and recipient may be both UCP and SCP, which in turn generates four different communication scenarios. Communication in each of these scenarios can be synchronous or asynchronous. We call *synchronous* communication those instances of communication where the sender can deliver messages to the recipient without explicitly storing these messages in the channel. Otherwise, we refer to *asynchronous* communication when messages sent by the sender may

need to be stored by the channel before being delivered to the recipient. If the channel stores the message we say that we are dealing with an asynchronous communication scheme. Therefore, if the channel discards the message when the recipient is not active, we say we are dealing with a synchronous communication scheme.

In an asynchronous communication scheme there are 2 options for the delivery of stored messages: (a) the channel attempts delivery to recipients, either when the recipient is active or through recurrent retries, or (b) the recipients ask the channel for messages (producer/consumer model).

The type of communication (synchronous or asynchronous), and the type of sender and/or recipient (SCP or UCP), give place to eight communication scenarios in groupware systems. These scenarios are shown in the taxonomy that appears in Figure 1.
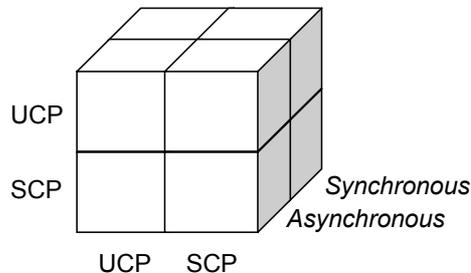

Figure 1. Communication taxonomy for groupware systems

To define this taxonomy the whole set of recipient processes has been considered as a homogeneous group either composed of SCPs or UCPs. The scenarios that use heterogeneous groups of recipients are very scarce and can be modeled by using two communication instances with homogeneous groups. The following two sections describe the four synchronous scenarios, and the four asynchronous scenarios.

The following section describes an architectural pattern (Buschmann, 1996) that may support the design and construction of communication mechanisms in collaborative applications. Then, section 3 states the conclusions of our work.

2 THE CSGS PATTERN

Collaborative applications need to manage communication between processes in any of the above scenarios. Building mechanisms that make this

possible is not an easy task, for up to this moment there are no guides or patterns to support the design of communication features in groupware systems. For this reason, we propose the CSGS architectural pattern as a support tool for the design and construction of communication processes for collaborative applications. This pattern is the result of our experience in developing collaborative applications. Next, CSGS is described using the pattern structure proposed by Buschmann [1].

2.1 Pattern Name

CSGS - Communication Scenarios for Groupware Systems.

2.2 Context

In a groupware system both the collaborators and some system processes need to exchange messages among themselves. In addition, sometimes it is necessary to keep a record of messages in order to subsequently operate on them. The types of communication that may be found in the collaborative systems are point-to-point, broadcast and/or multicast. At the same time, this communication can be synchronous or asynchronous.

2.3 Problem

Usually, any groupware application consists of modules that are executed in different machines and possibly at different times. These modules contain processes that send messages to processes contained in other modules. The system should guarantee that these messages reach the proper recipient. At times, it is necessary to store these messages for subsequent operations, such as queries.

Several situations can arise. For example, a user could explicitly send a message to another user or user group; an internal process could send a message to other process(es), or to other user(s); a user --or process-- could receive messages at any moment; a user or module in the system could request reviewing previously delivered messages. Some messages should be displayed, in such a way, as to allow the user to interpret them.

2.4 Solution

In order to manage the above communication cases, a groupware application should consist of at least a work interface, an event handler, and a channel. It may also require one or more system processes in charge of performing tasks to support collaborative work; for example a session manager or a floor-control manager, among others.

The *user interface* is a process that allows the display of messages that should be interpreted by the user. The *event handler* has two functions: (a) it is in charge of receiving messages and displaying in the user interface, those messages whose recipient is a UCP, and (b) it is in charge of capturing the events that should be transmitted, and sending them through the channel to the proper recipient processes. The *channel* is to take care of the transportation of messages between the event handlers. The channel should have three associated elements: a *session manager*, a *log file*, and a *buffer*. The channel needs to manage the sessions, and this is performed through the work session manager, which saves information about the active sessions, the users connected to each of them and the shared objects (Guerrero, et. al, 2001). Every message that travels through the channel is properly stored in a *log file*.

2.5 Structure

CSGS comprises four types of participating components: processes, user interface, event handler, and channel. In turn, the channel has three additional components: buffer, log file, and session manager.

The *processes* can be UCP or SCP. The *user interface* allows the user to create messages to be transmitted and also to display the messages, which the user receives so that he or she can interpret it. The *event handler* captures the events generated in the interface and sends to the interface the messages that should be displayed for the user. The *channel transmits* the messages. The *buffer* associated to the channel allows the storage of messages for a subsequent delivery. The *log file* maintains all the work history. Finally, the *session manager* recognizes the users that are working in each session.

Messages sent by system processes may be a coordinate in a drawing area, the blocking of a shared object, the arrival a new user into a work session, a command, etc. These messages are used in applications such as shared whiteboards, telepointers, etc. From the point of view of the channel, messages are simply "objects to be sent", regardless if they are generated by a user process, or by an internal system process. Also, it does not matter if they follow an event notification, an RPC command, etc. Figure 2 shows the object class diagram (UML) that conforms the CSGS pattern.
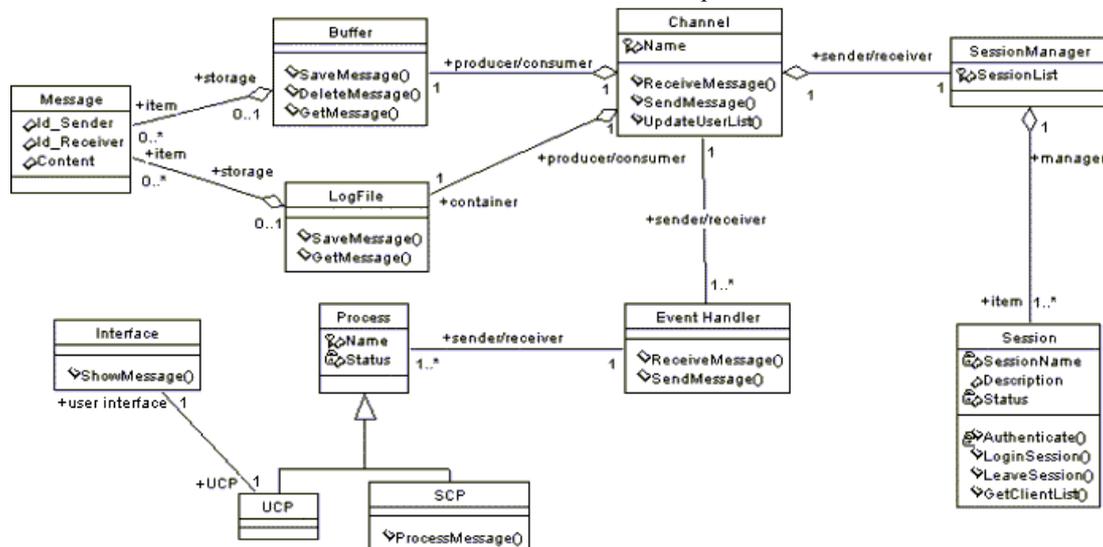

Figure 2: Object classes diagram for the CSGS pattern

2.6 Implementation

1. It is necessary to define the required functionality of the pattern. It must be decided if the log file is wanted or not, as well as the storage time for the messages in it. For example, last hour, last week, last 100 messages, etc.
2. It is necessary to decide whether the communication is synchronous or asynchronous, in order to decide if the *buffer* component is to be used or not.
3. It is necessary to specify what messages can be generated both by user and by internal processes. It is also necessary to specify what messages can arrive at each user or process, and what to do with them. Messages that are received by users should be displayed by the *interface*.
4. It should be decided whether to use a client-server or a point-to-point architecture. The decision made will determine where to store the list of users logged into the work session, as well as their corresponding IP.
5. If a *buffer* or a *log file* is used, it is necessary to define where they will be implemented, and where to store the messages. For example, in client-server architecture, these two components, as well as the *channel* component, can be implemented as part of the main server, or they can be implemented as independent servers.

2.7 Known Uses

Several variations of the CSGS pattern are present in most frameworks for groupware application development, such as: Groupkit, Habanero, JSDT (Java Shared Data Toolkit), TOP (Ten Object Platform), among others. Also, the pattern can be found in technologies of distributed objects such as: CORBA, DCOM, and EJB. This indicates that CSGS can be used for the development of groupware systems, as well as the creation of frameworks for the creation of such systems.

3 CONCLUSIONS

This paper presents not only a taxonomy to show the communication scenarios that can be found in groupware systems, and a pattern to handle these scenarios, but also a way to organize computer-mediated communication in order to easily implement the design aspects that depend on this communication. These aspects include: awareness; process coordination; notification; and user, session and floor control management. Many of these aspects are critical for the success or failure of a collaborative application.

In relation with it, the CSGS pattern provides some important benefits as: (a) abstract all communication aspects across networks, (b) independently implements components, (c) sends any type of message through the same channel, (d) permits the implementation of synchronous and asynchronous communication between processes, and (e) maintains the history of the communication process.

ACKNOWLEDGEMENT

REFERENCES

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. Stal, S., 1996. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons.

Guerrero, L.A., Fuller, D., 2001. A Pattern System for the Development of Collaborative Applications. *Information and Software Technology*. Elsevier Science B.V., 43, 7, (May 2001), 457-467.

Miranda, H., Rodrigues, L., 1999. Flexible Communication Suppport for CSCW Applications. In *Proceedings of 5th International Workshop on Groupware (CRIWG'99)*. Cancún, Mexico, (September 21-24, 1999), 338-342.