Licea, Guillermo; Favela, Jesús
Reusability in Groupware Development through a Pattern System
Computación y Sistemas, vol. 3, núm. 3, enero-marzo, 2000, pp. 182-191
Instituto Politécnico Nacional
Distrito Federal, México

Available in: http://www.redalyc.org/articulo.oa?id=61530304
Reusability in Groupware Development through a Pattern System

Guillermo Licea and Jesús Favela
Departamento de Ciencias de la Computación
Centro de Investigación Científica y de Educación Superior de Ensenada
Km. 107 Carretera Tijuana-Ensenada, 22860, Ensenada, B. C., México
{licea, favela}@cicese.mx

Article received on June 14, 1999; accepted on December 3, 1999

Abstract

We present a pattern system named COCHI (Collaborative Objects for Communication and Human Interaction), which supports reusability in the development of groupware at three levels of abstraction: architecture, design and implementation. COCHI has two properties: ease of use and flexibility, but in contrast with other tools, COCHI maintains an equilibrium between both goals providing a class framework that allows a developer to implement groupware applications reducing the implementation effort and a set of architectural and design patterns that adds flexibility to the platform and permits its extension with new features. This makes COCHI particularly useful for the research of new multi-user interface widgets, coordination mechanisms, floor control policies, etc.

We show how COCHI’s class framework reduces the effort required to implement synchronous groupware and how the architectural and design patterns have been extended to support features like Quality of Service (QoS) and Emotional Awareness not present originally. Thus, we show how ease of implementation is not necessarily in opposition with extensibility in reusable software.

Keywords:
Reusability, pattern system, groupware application.

1 Introduction

Reusability has been one of the main topics in the software engineering discipline and one of the primary goals. Reusability is defined by the IEEE as “the degree to which a software module or other work product can be used in more than one computing program or software system” (Institute of Electrical and Electronics Engineers, 1990).

According to McIlroy (McIlroy, 1968) the basic idea behind software reuse is very simple: “Develop systems of components of a reasonable size and reuse them”. The next step consists in extending the concept of “systems of components” beyond the implementation or coding phase to analysis, design, etc. because reusability can help developers in every phase of the development process and reduces the redundant work enhancing the reliability of the system because each piece reused was already tested.

Reusability can be achieved, basically, at three levels of abstraction: architecture, design, and code, which correspond to high level design, detailed design, and implementation phases of the software development process.

The architecture of a software system defines the structure, interfaces, and interaction between the parts (components, subsystems, etc.) of the architecture. As defined by Jacobson (Jacobson et al., 1997) architecture reuse can be achieved when “in an object-oriented system where implementation classes are organized in subsystems, the software architecture defines the static organization of software into subsystems interconnected through interfaces and defines at a significant level how nodes executing those software subsystems interact with each other”.
One well-known approach to build a software architecture to be reused is to organize it in several layers. A layered architecture allows us to organize software from its most general to its most specific. In a layered architecture subsystems are organized in layers, where each layer is supported by a more general layer (Shaw and Garland, 1996). Figure 1 shows a typical layered architecture.

Figure 1: A typical layered architecture

The applications layer includes variants of particular applications. The domain specific layer contains components that implement functionality shared by their applications. The middleware layer includes platform independent components. And finally, the system software layer contains interfaces for the actual hardware infrastructure.

Reuse at the design level involves the use of a predefined skeleton for modeling some particular feature, normally related to a subsystem of an application’s architecture. The best known technique for reusing design components or packages is the Design Patterns. Design patterns are documents that include the context, problem description, restrictions and a good solution to a problem. Normally this solution is given in the form of a set of diagrams.

When we combine design patterns with software components we generate an application framework (Johnson, 1997), which is defined as “a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact”.

Software reuse has traditionally been achieved at the level of code. The most common technique to code reuse consists in the development of software based on libraries of functions, procedures, or classes. Other technique for code reuse consist in the use of visual programming environments for specific domains such as user interfaces and data bases among others. These environments provide an application skeleton for a well defined design space.

As we have seen in this section, there are several techniques and tools that allows us to reuse at any of the three different levels of abstraction. In this paper we present COCHI, a groupware pattern system which includes reusable components at the three levels. Groupware is defined as a computer-based system that supports a group of persons to achieve a common task providing a shared interface (Ellis et al., 1991). Groupware applications include features related to collaboration, coordination, and communication that make them complex to design and implement.

There are several groupware development tools. Some of them are easy to use, but not flexible. Others are very flexible, but hard to use. COCHI, as a pattern system, combines and maintains in equilibrium these two important features of development tools, ease of use and flexibility.

In the next section (section 2) of this paper we discuss about the object-oriented reuse technique known as pattern systems to understand the ideas behind COCHI. In section 3 we show the details of the COCHI pattern system. In the section 4 we illustrate the tradeoff between ease of use and flexibility that makes COCHI an extensible groupware development tool. In section 5 we show our conclusions. And, finally, in section 6 we propose future work.

2 What is a Pattern System?

The concept of pattern can be traced to Christopher Alexander (Alexander, 1977) who wrote: “Each pattern describes a problem that occurs many times in our environment and then describes the most important part of the solution to this problem, so we can use this solution many times”. Alexander was talking about buildings, but what he said applies to software too.

Software patterns are based on the experience of software developers and can be generated from abstractions of the generic characteristics of representative systems in a specific area. Software patterns provide a systematic approach for the reusability of code or design packages (classes, modules, etc.). There are several specialized catalogs of design patterns (Coad et al., 1995; Fowler, 1997; Gamma et al., 1995), pattern languages (Coplien and Schmidt, 1995; Martin et al., 1997; Vlissides et al., 1996), pattern systems (Buschmann et al., 1996; Keller et al., 1998), and related work that shows the importance and success of patterns in software development.

There are basically three types of software patterns: architectural patterns, design patterns, and idioms. An architectural pattern defines the structure and organization of a particular type of system. It shows the big picture of the system including the subsystems and the relationships between them in the defined architecture. A design pattern shows the details of a subsystem and documents a framework. A design pattern contains the design rationale behind a framework what makes it easier to extend or modify. An idiom is a low-level pattern that describes how to implement particular aspects of components using a specific programming language. We can say that a framework is generated by the classes implemented by idioms and documented by design patterns.
A pattern system is composed of a set of software patterns, including architectural patterns, design patterns, and idioms. A pattern system describes how its constituent patterns are related with other patterns, how these patterns can be implemented, and how can software development be supported using the patterns.

Figure 2: Relationship between architectural pattern, design pattern and class framework

Figure 2 illustrates the relationships between the elements of a pattern system and it shows how a class framework can be benefited with the use of architectural and design patterns to generate a pattern system.

In the next section we present COCHI, a pattern system built by the authors, which was created to support reuse during the development process of groupware applications.

3 COCHI, a Pattern System for Groupware

Groupware is defined as a computer-based system that supports a group of persons to achieve a common task providing a shared interface (Ellis et al., 1991). Groupware applications include features related to collaboration, coordination, and communication that make them complex.

In (Favela et al., 1997) we proposed a design space for groupware with seven dimensions obtained from the analysis of eleven groupware applications found in the CSCW literature. The dimensions are: interaction, participation, visualization, notification, coordination, distribution, and support.

Considering the design elements of each dimension of the groupware design space, we found that these can be classified in six groups that will be explained in the next section as the subsystems of the COCHI architecture: session management, floor control management, collaboration awareness, communication media, objects management, and collaborative interface.

The COCHI pattern system consists of a set of software patterns and Java classes that can be used to design and implement synchronous groupware applications, in which the participants interact with each other at the same time. Next we present the three components of the COCHI pattern system: the architecture with all its subsystems, the description of each subsystem and the implementation of the classes included in the subsystems.

3.1 The COCHI Architecture

The first element of the COCHI pattern system is a layered architecture composed of several subsystems that can be described as design patterns. We use layers because they reduce software dependencies and promote reusability (Jacobson et al., 1997). Figure 3 illustrates the architecture of the COCHI pattern system.

The architecture itself is based on a design pattern (Client-Server design pattern) that allows the participant’s applications to communicate with each other through a server with a defined protocol.

The server is composed of three layers. The configuration layer allows participants to configure the characteristics of the session. The collaboration layer of the server includes five subsystems that extend the Client-Server design pattern included in the server layer.

The participant is composed of four layers. The application layer is the groupware application itself. The collaborative interface layer represents the interface between the developer and the collaboration layer, its features and services. This layer includes the necessary elements for developing groupware applications based on the COCHI platform. The collaboration layer of a session participant has the same five subsystems included in the collaboration layer of the server, so each subsystem of the participant interacts with the corresponding subsystem of the server. The participant layer contains the Client-Server design pattern.

Figure 3: Architecture of COCHI
3.2 The COCHI Subsystems

In this section we describe the subsystems included in the server, participant and collaboration layers of both server and session participant. We begin with the description of the Client-server design pattern and then we present the five subsystems included in the collaboration layer.

The Client-Server Design Pattern

The Client-Server design pattern includes the classes needed for handling the connection and event passing between groupware applications through a server. Figure 4 shows the structure of the classes included in the design pattern.

The Server class is a thread (concurrent object) that defines a generic server that listens through some port and receives connection requests from participants. The server creates an instance of Connection handler for each connection requested and uses this instance to receive/send events from/to a particular participant.

The Server subsystem is a specialization of server that receives and processes events sent by the participants connected to the server and distributes the events to some or all the participants connected.

The Client class is a thread that defines a generic client that requests connection to the server and it receives/sends events from/to the server once the connection is established.

The Client subsystem is a specialization of participant that receives the events sent by the server and processes them according to the functionality of the collaborative feature it implements. Client subsystem can also send any type of event defined for the application to the server to be distributed to other participants connected.

The Session Manager Design Pattern

The session manager design pattern is a specialization of the Client-Server design pattern. The subsystem provides functionality that allows participants to join and exit from a particular session. This subsystem also registers the session name and information from each participant (identification, name and role). Figure 5 presents the classes included in this subsystem and the design patterns associated with it.

In Figure 5 we use the name Collaborative design pattern to indicate any of the design patterns included in the collaboration layer that will be explained in the next subsections and Collaborative interface refers to the subsystem contained in the collaborative interface layer of the session participant.

The Communication Media Design Pattern

The communication media design pattern is a specialization of the Client-Server design pattern. The subsystem allows participants to communicate with other participants in the session using one or more media: text, audio, or video.

Figure 6 shows the classes included in this subsystem and the design patterns associated with it.

The collaboration Awareness Design Pattern

The collaboration awareness design pattern is a specialization of the Client-Server design pattern. It defines a subsystem that allows participants to be aware of the presence of other participants in the session. Participants are represented using a list, labeled pointers, icons or, a status line. Figure 7 shows the classes included in this subsystem and the design patterns associated with it.
The Object Manager Design Pattern

The object manager design pattern is a specialization of the Client-Server design pattern. It defines functionality for participants to save objects, or groups of objects in a repository included in the server and to retrieve these objects in a latter session. Figure 8 shows the classes included in this subsystem and the design patterns associated with it.

The Floor Control Design Pattern

The floor control design pattern defines a subsystem to coordinate tasks between session participants through a defined floor control policy using one or more floor control mechanisms. The pre-defined mechanisms are: free, random, pre-emptive, FIFO, central moderator, pause detection and round robin. Figure 9 shows the classes included in this subsystem and the design patterns associated with it.

The Collaborative Interface Subsystem

The collaborative interface subsystem is included in the collaborative interface layer of the participant, it is composed of different types of windows and collaborative components. The purpose of this subsystem is to support the developer in the design of the user interface of a groupware application.

There are two groups of elements contained in the collaborative interface subsystem, specialized windows and collaborative user interface components or widgets. Collaborative, private and secondary windows are specialization of Window, a generic window associated with the session manager subsystem.

Collaborative window is the main window of a groupware application. It is a public window that automatically sends to the server all the events generated in it. This window also receives all the events generated by other participants in the session. A Secondary window is a public window that can be used by a collaborative window to perform specific tasks. The Private window stores all the events generated inside it until the user decides to send them.

Figure 10 shows the classes included in the subsystem and the way in which they relate to the design patterns shown in the previous sections.

The Class Framework

The class framework of COCHI implements, in the Java programming language, all the classes included in the design patterns and subsystems adding some useful classes that facilitate the implementation of groupware applications.
Following the architecture, the framework is divided in a client and a server class hierarchy. Programmers do not need to modify or extend any class in the server part of the framework. The framework contains a class to configure the server before its execution.

To implement a groupware application or convert an existing single user application using the framework defined in COCHI, a programmer needs to specialize the methods defined in a few classes, the ones included in the collaborative interface subsystem contained in the participant.

COCHI’s class framework has been used to implement more than ten groupware applications such as: collaborative painting, CRC cards generation (Navarro et al., 1997), house of quality method (Ortiz et al., 1997), gesture recognition for collaborative systems (Contreras et al., 1997), among others.

Table 1 shows a brief description and characteristics of some of the applications implemented with COCHI’s frameworks. Developer’s skills refers to the experience using the Java programming language and COCHI’s framework. Novice has no experience with Java or COCHI, medium has some experience with Java, but not with COCHI, and expert has experience with Java and COCHI. Time represents the number of hours dedicated to the application, and lines of code refers to the code written by the developers, not including the code provided by COCHI’s framework.

<table>
<thead>
<tr>
<th>Name/ Description</th>
<th>Developer’s skills</th>
<th>Time in Hours</th>
<th>Lines of code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Paint</td>
<td>Medium</td>
<td>1</td>
<td>41</td>
</tr>
<tr>
<td>Free hand drawing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paint</td>
<td>Expert</td>
<td>16</td>
<td>239</td>
</tr>
<tr>
<td>Drawing of figures such as</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lines, circles, etc. using a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set of colors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HQ</td>
<td>Novice</td>
<td>24</td>
<td>222</td>
</tr>
<tr>
<td>Generation of product design</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>attributes based on customer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>needs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agenda</td>
<td>Novice</td>
<td>80</td>
<td>485</td>
</tr>
<tr>
<td>Brainstorming and decision</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>support</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gesture recognition</td>
<td>Novice</td>
<td>62</td>
<td>754</td>
</tr>
<tr>
<td>Gesture recognition and</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>annotations over documents</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTO</td>
<td>Medium</td>
<td>70</td>
<td>1481</td>
</tr>
<tr>
<td>Generation of CRC cards that</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>describe classes during</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>object-oriented analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

These applications have been implemented mostly as final projects in a graduate course on collaborative systems and we found that students using COCHI’s class framework were able to focus on the application’s functionality rather than low-level issues like: how to send/receive information to/from session participants? or what kind of floor control policy and mechanism must be used?, among others. It was remarkable that the complexity of the applications developed using COCHI was much higher than those developed without it.

4 Extending COCHI’s Design Space

In Figure 11 we have classified different types of software development environments according to ease of use (to what extent they simplify application development) and flexibility (the facility with which new functionality, not originally considered, can be incorporated). A programming language for instance, provides a great deal of flexibility, but is not particularly easy to use. At the other extreme an application provides little flexibility to support functionality not originally considered, but requires no development effort.

The ideal development environment will be one that provides maximum flexibility and ease of use. We can note in Figure 11 that a pattern system comes close to this ideal, since it provides ease of use without compromising flexibility. It reduces implementation effort since the patterns document the class framework and permit to extend them for specific applications or families of applications.

A pattern system is not as easy to use as a final application or a run-time support platform, but clearly a pattern system is easier to use than a class framework because design patterns document the class framework and permit to extend them for specific applications or families of applications reducing the implementation effort. Design patterns can be considered as the micro-architectural elements of a framework (Beck et al., 1994; Johnson, 1997).

There are many tools that support the development of groupware applications such as class frameworks (Burridge, 1998; Chabert et al., 1998; Dewan, 1991; Roseman and Greenberg, 1996; Schuckmann et al., 1996), libraries (Winnet et al., 1994), run-time environments (Crowley et al., 1990; Patterson et al., 1990), visual environments (Banavar et al., 1998) (among others), but they were built to support the implementation of groupware applications under some design space and they lack the flexibility needed to be extended to support new features.

In contrast with other solutions, the structure of COCHI based on software patterns, allows designers of groupware to customize the platform according to their particular needs. The architecture, documented through an architectural pattern, can be extended with new subsystems and each of which
(design pattern) can be extended with different features, in addition to those already supported.

Figure 11: Ease of use vs. flexibility provided by software development environments

In the previous section, we saw that the class framework of the COCHI pattern system provides ease of use to implement groupware applications and we described briefly some of the groupware applications implemented. In this section, we present two extensions made to the COCHI pattern system to illustrate this argument that support adaptive Quality of Service and Emotional Awareness in groupware applications. In both extensions it was necessary to work in the three levels of abstraction supported by COCHI. The COCHI architecture was modified to add new subsystems, each subsystem was specified as a design pattern using the already defined design patterns, and finally, were implemented as Java classes to extend the COCHI framework with the new features.

4.1 Adaptive Quality of Service in Groupware Applications

Quality of Service (QoS) is a concept used in network applications to addresses performance guaranties, so that applications can specify their requirements and the network will try to satisfy them. Writing applications to take advantage of the QoS concept is not a trivial task. Developers have to pay attention to details such as building the appropriate data structures to make a QoS specification, passing the right arguments to the right functions, selecting the right API and learning all its idiosyncrasies.

The extension made to COCHI allows groupware application developers to work at a higher level of abstraction, overcoming unnecessary complexity and avoiding common pitfalls that arise when using low-level native OS APIs (Licea et al., 1997). In this section we describe the subsystems included in the COCHI extension.

The QoS extension adds three QoS subsystems to COCHI. The subsystems include features such as QoS management, QoS specification, and QoS-aware communication media.

QoS management involves a QoS manager (based on the Client-Server design pattern) for three collaborative scenarios: casual interaction, brainstorm session, and virtual classroom. QoS specification is associated with the QoS manager and considers different types of network infrastructures such as ATM, RSVP, among others. Figure 12 shows the classes and relations between them and the session manager subsystem.

Figure 12: QoS manager and QoS specification

QoS-aware communication media adds some classes to the Communication media design subsystem included in COCHI to support QoS-aware video and audio (Figure 13).

Figure 13: QoS-aware communication media

Figure 14 shows a casual interaction scenario using QoS-aware video as communication media. With this application a virtual lab environment could be created where two participants in different locations have a video camera to feed a video reflector. When a participant connects to the reflector, it will take the video feed from the other location and display it in the video window, so there will be awareness of what is going on at the counterpart. In this awareness mode the QoS level for the video window will be set to high.
4.2 Emotional Awareness for Groupware Applications

Affective computing is a computer science research field that relates to, arises from or deliberately influences human emotions (Picard, 1997).

This extension to COCHI allows the development of applications that provide awareness of the emotional state of participants (Garcia et al., 1999). This knowledge can help improve decision making in collaborative activities. For example, participants could notice the level of understanding, stress, frustration, or satisfaction of other participants and act accordingly. Figure 15 shows the new classes integrated to COCHI to support emotional awareness and their relationship to other components.

Figure 15: Emotional awareness

*EA-icon* (Emotional Awareness icon) is an extension of the *Icons* class used to represent user participants through icons. *EA-icon* represents the emotional state of each participant in a collaborative session. The *EA-graph* is a window displaying a graph that uses a three-dimensional space used to describe and measure emotional states. *Affective engine* is a concurrent object that detects user emotions using different kind of devices.

Figure 16 shows an enhanced version of the Use Cases drawing tool application, developed before, with an emotional awareness widget. We used this tool to apply a test to a set of students in order to measure their practical knowledge in Use Case diagramming. In this case, the affective awareness facility serves as a communication media where students look for clues to acknowledge their performance. In this application we use the *EA-graph* to represent the affective state of each participant collaborating. In the Figure we can see: a) Chat window, b) Sketching window, c) Video conferencing window, and d) Emotion awareness graph.

5 Conclusions

Reusability has been one of the main topics in the software engineering discipline and one of its primary goals. Reusability can be achieved at three levels of abstraction: analysis, design, and implementation. However, reusability at higher levels during the software development process could bring more benefits than the ones offered by reusability during coding or implementation only. At the code level, we can reuse, for instance, the functions or classes provided by a library, but these functions or classes were implemented in the context of a design space defined by this library, so we are restricted to this space. When we reuse at the design level, we can add new features to the design space and they will be reflected in the code, thus facilitating the extension of the design space they define.

Software patterns have been proposed for the development of several kinds of systems, including distributed and network systems (Keller et al., 1998; Van den Broecke et al., 1997), but, none of these patterns deal specifically with groupware. On the other hand, class frameworks, libraries, and toolkits have been developed specifically for groupware, but they were built to support the implementation of groupware applications and define a design space that, as we mentioned, lack flexibility to extend the design space with new functionality.
We show how the reusability technique named pattern systems can be applied for the reuse at architecture and design levels, in addition to programming. We illustrate this through the use of COCHI, a pattern system that facilitates the development of groupware applications. COCHI includes an architectural pattern that provides a general structure for the fundamental features of a groupware application represented as subsystems in the architecture. COCHI also provides a set of design patterns that document each subsystem, and at the lower level of abstraction COCHI defines a Java class framework that implements all the classes included in the design patterns.

Concluding, we can say that the COCHI pattern system provides a good basis for reusability at the three levels of abstraction due to its ease of use and extensibility. We have shown that COCHI’s class framework provides an easy to use platform to implement groupware and COCHI’s architectural and design patterns provide extensibility for new functionality such as Quality of Service and Emotional Awareness features mentioned in section 4 that can be easily extended to all applications developed in COCHI.

6 Future Work

As future work we propose the automatic generation of groupware application templates. We are currently building a toolkit that allows to select from several options in the design space and generate a groupware application template according to the requirements of the system and design decisions taken. The developer should complete the application according to the functionality required. In addition, we also propose a visual environment that will support the implementation of the multi-user interface of the groupware application.

We expect COCHI to evolve with new extensions to the architectural and design patterns such as new kind of collaboration awareness, support to multiple sessions, and new subsystems to support useful features like mobile agents that participate during a groupware session and nomadic computing to allow the use of different kind of computing devices such as Personal Digital Assistants, palmtops, and laptops during a session.

Finally, we are currently porting COCHI’s class frameworks to support the implementation of groupware applications running over the WWW (World Wide Web), the ubiquitous, largest communication platform.

References


