A Design Pattern for Representing Workspace Awareness

Miguel A. Teruel, Elena Navarro, Víctor López-Jaquero, Francisco Montero, Pascual González
LoUISE Research Group
University of Castilla – La Mancha
Albacete, Spain
{miguel, enavarro, victor, fmontero, pgonzalez}@dsi.uclm.es

Abstract— In the same way as it happens when designing a one-user system, designers and developers will probably have to address some design problems that have appeared often previously in some other developments. This kind of recurrent problem can be identified and captured in a design pattern that provides a reusable solution to a common problem. The main aim of this work is to provide a design pattern that can be applied to the design of CSCW applications supporting Workspace Awareness (WA). Hence, this object-oriented pattern answers all WA questions by means of a class structure representing the typical entities and behaviors available in collaborative applications (e.g. users, artifacts, collaborative actions). Moreover, a proof of concept is presented, thus assessing the feasibility of this pattern with a concrete technology, namely C# and Visual Studio 2012.

Keywords— design pattern; Workspace Awareness, Visual Studio

I. INTRODUCTION

A design pattern is a reusable solution to a recurrent problem appearing when designing software. Nevertheless, the use of patterns did not first appear in Computer Science, but in Alexander’s et al. book on architecture, urban design, and community livability [1]. In this book, the authors presented 253 patterns answering questions like “How high should a window sill be?” or “How much space in a neighborhood should be devoted to grass and trees?”, thus creating what the authors call “a pattern language”. Thereby, one of the main advantages of using design patterns is that of trying to solve a design problem which has been previously identified in a design pattern. There is no use in making up our own solution when there is a widely accepted and validated pattern which provides a proper design solution to our problem.

The patterns issue can be extrapolated to CSCW field, whose users usually have common behaviors that can be captured by means of design patterns [2], [3]. For example, when performing a task that involves collaboration, it is usual that several users perform collaborative actions over an artifact whose state will be modified as a result of this action. Thus, by providing a design pattern collection rich enough we can contribute to solve an important amount of collaboration “problems” by reusing well-known solutions.

Another problem that arises when designing a collaboration system is how to deal with the users’ awareness information needs. Briefly, in order to participate in a CSCW system, users must be aware of the other users with whom to perform collaborative tasks. That means the users should be provided with information about who are available to collaborate with, what the other participants are doing, where they are working on, when they performed an action or how an artifact got certain state. In order to answer these questions, as well as several others, Gutwin presented his awareness interpretation, Workspace Awareness [4], a framework describing all the elements, regarding both present and past, that must be considered when providing awareness to a CSCW system’s users.

Since WA is considered one of the most widely-accepted awareness interpretations for groupware, many collaborative applications implement awareness features by answering these WA questions. Because of that, we present in this work an object-oriented design pattern supporting the definition of all the concepts and questions underlying WA. Nevertheless, other awareness interpretations are available in the literature [5], [6] that could also be used.

The WA pattern here presented can be reused in the design of a new collaborative system which needs to support all (or some of) the WA elements. In addition, this pattern facilitates its adaptation to different systems by means of several class hierarchies which in turn, can be extended in order to support a more specific environment. In addition, a proof of concept of the pattern is presented in this work implemented by using Visual Studio 2012 as development environment and C# as programming language.

This paper is structured as follows. Just after this introduction, some related work is discussed en Section II. Next, the object-oriented design pattern supporting WA is fully described. Then, the proof of concept that demonstrates the feasibility of the pattern presented is explained. Finally, some conclusions and further work are shown in the last section.

II. RELATED WORK

There is plenty of literature about design patterns in general. For instance Gamma et al. presented a collection of object-oriented design patterns aiming at solving specific and recurrent problems in software design [7]. This collection of creational, structural and behavioral patterns helps designers to reuse well-known solutions, thus reducing both development time and design errors.
As far as CSCW is concerned, an important amount of work has been done in order to elaborate collections of patterns. For instance, Schummer and Lukosch guide the design of Human-Computer-Human Interaction by providing a collection of patterns supporting communities and groups of users [2]. Thus, by using a pattern language they propose an important amount of patterns focusing on the human user of the system as well as assisting developers in the development process of CSCW systems. Another work, focused on CSCW was presented by Isla et al., consisting on the definition and application of patterns to model social organizations [8]. In this work, the authors proposed a template for describing the social organization of the members, participating in collaborative processes. This technique, which models patterns by using an extension of UML state machine diagrams, was included in AMENITIES [9], a methodology aimed at the analysis, design and development of CSCW systems.

A different pattern collection is proposed by Vreede et al. in [10]. In that work, the authors propose 60 predictable and repeatable patterns, called ThinkLets, aimed at specifying Group Support Systems (GSS) and mapping seven basic patterns for thinking: diverge, converge, organize, elaborate, abstract, evaluate, and build consensus. Moreover, these patterns are supported by a tool [11] that guides the design of GSS by using those Thinklets. Another work of an empirical nature regarding patterns for CSCW was presented by Iacob [12]. In this study, she evaluated the impact of using a collection of CSCW patterns [13] on the design of synchronous applications by evaluating 18 students’ teams. Moreover, by means of that experiment, the understandability of those patterns was measured as well as the strategies that novice designers developed when working with that collection of design patterns.

However, as far as we know, there are no design patterns providing a reusable solution for applications with WA support. Moreover, taking into account the fact that WA is considered the most widely-accepted awareness interpretation for groupware development, a design pattern that answers the different WA questions can be considered an interesting contribution, which constitutes the main aim of this work.

III. A DESIGN PATTERN FOR WORKSPACE AWARENESS

In this section the pattern that supports WA is presented. Thus, by implementing this pattern (see Fig. 1), the developed CSCW can support the required WA elements identified by Gutwin more easily. Moreover, this pattern provides the basic class structure that a collaborative application should have, that is, a set of users who will perform whether individual or collaborative actions over artifacts (whose state will be modified by these actions) in order to satisfy a goal.

This pattern consists of several classes related among them by inheritance and association relationships which will enable the implemented CSCW applications to provide its users with awareness information about the other users with whom to collaborate. Concretely, the classes that are part of this pattern are the following:

- **User**: this is the class representing an entity (human or computational) who is participating in the collaborative system. It has several methods that answer to some of the WA questions. For instance, the method `whatIsWorkingOn()` is answering to the question “What object are they working on?” for the present/what/artifact category. It is worth noting that this pattern distinguishes two different types of past: the recent past and the distant past. The reason for having these two types of past is that in a CSCW applications the users are interested in what their collaborators are doing or what they have recently done, being the information regarding the distant past actions hardly ever queried. Because of that, this class has, for instance, the method `whatHasBeenDoing()` that returns the last actions performed by the user (i.e. in the last session, thus representing the recent past), and the method `whatDid(from, till)` returning the actions performed in the past between two dates (representing the distant past). The rest of WA questions regarding the users are represented in the pattern by means of association relationships. For example, the association between `User` and `Location` answer the question “Where are they working?”

- **Action and KindOfAction**: the class `Action` represents an action performed by a user on an artifact and, optionally, aimed at satisfying a goal. By including in the Action class a field representing the time when the action was performed, this class is able to answer the question “When did that event happen?”. Note that in this pattern the term “event” appearing in WA question has been replaced for the term “action” in order to avoid the confusion between Gutwin’s events and programing events (software pattern in which a notification source makes callbacks to one or more handler methods). In addition, this class has association relationships with `Goal` and `User`, to answer the questions “What goal is that action part of?” and “Who is doing that?” respectively. The class `KindOfAction` represents the different kinds of actions that can be performed in the system. For instance, in a collaborative model editor, the kinds of actions could be: adding an element, adding a relationship, modifying an element, removing a relationship, etc. Therefore, an `Action` can be considered as an instance of a `KindOfAction`. Moreover, `KindOfAction` can be refined when necessary into different sub-classes when special fields or methods are needed for specific kinds of actions.

- **Goal**: as aforementioned, an action can contribute to satisfy a goal. Consequently, this concept is represented by the `Goal` class which, by means of its association with the class `Action`, answers the WA question “What goal is that action part of?”, thus defining the intention of an action. Moreover, provided a Goal-Oriented Requirements Engineering technique, such as CSRM [14], has been used to specify the system requirements, this class can be directly traced to the goal concept of the requirements model.

- **Artifact and State**: the Artifact class represents the domain elements that the users will perform their actions on. This class answers to the WA question “What object are they working on?” with a method returning an array
of User corresponding to the user(s) who are working with this artifact. In addition, as previously discussed for the User class, two different types of past are considered for the Artifact class. The former offers information about the artifact history in the near past; and, the latter provides further historical information bounded by two dates. Furthermore, the different states that an artifact has had are represented by means of instances of the State class. Each instance has information about when and how (by means of what action) the artifact come to be in that state.

- **Position and sub-classes**: the Position abstract class can represent either where in the workspace an artifact is positioned or where a user is looking at. Due to the wide variety of CSCW systems, this class must be specialized in a class representing the positioning mechanism of a concrete system. For instance, we propose in this pattern three of them corresponding to a two-dimensional position of a typical WIMP system (Position2D), a three-dimensional position of a virtual reality workspace (Position3D) as well as the position where the user is looking at (Cursor).

- **View and sub-classes**: the View abstract class is used to answer “Where can they see / reach?”. For this class, on one hand, we use an specialization for WIMP environments corresponding to a bounding rectangle, that is, the part of the whole workspace that the user is viewing (or can reach if he/she does not have access to the complete workspace) which, in this specific case, is represented by a view rectangle. On the other hand, the View class is also specialized in a tridimensional bounding box in order to make the pattern able to deal with 3D virtual environments. Nevertheless, the View class can be specialized in different view/ reach mechanisms when dealing with more complex systems (i.e. augmented reality environments).

- **Location and sub-classes**: when the user’s context information is relevant to the collaborative application, these classes must be taken into account. With this aim, the location of a user is represented by means of the Location abstract class. However, in a similar way to the Position class, a specialization is needed with regard to the system that the users are collaborating in. For instance, a GeographicalLocation should be used when it is necessary to manage the physical location of the user on Earth and a VirtualLocation when users have a virtual representation instead of a physical one. Moreover, these classes can be re-specialized if additional information is needed.

- **Management**: the class Management is responsible for managing the workspace (users, artifacts, actions). This class implements the methods isAnyone(), whoIsParticipating() and whoWas(from, till) answering to the WA questions “Is anyone in the workspace?”, “Who is participating?”, “Who is that?”, and “Who was here, and when?” respectively. Nevertheless, although only one management class appears in the pattern, it should be decomposed into several management classes (e.g. UsersManagement, ArtifactsManagements, etc.) in order to avoid the implementation of the design anti-pattern “the blob” [15].

---

![Fig. 1. Workspace Awareness design pattern](image-url)
In order to exemplify the use of this pattern, a concrete CSCW application, namely Age of Empires Online (AoEO, [18]) is used. This is a collaborative massive-multiplayer-online strategy game, whose users combat collaboratively with their allies in order to defeat their foes. Concretely, Fig. 2 shows a radar view [19] used in this game to provide the user with information about its environment. In order to put into practice the pattern, the designers of the AoEO would have to customize the elements of the pattern according to their needs. For instance, two sub-classes of Artifact could be defined to represent the user’s bases and troops (blue icons). As Artifact is related to Position, and therefore to Position2d (see Fig. 3), each one of the new subclasses would have automatically this relationship. For instance, every instance of the subclass troop could be placed in the radar view thanks to this capability (see yellow icons in Fig. 2). As Artifact is related to State, the instances of the new subclasses could have information about their state, such as the remaining life of a soldier of a troop or the evolution age of a base. The class User could be also specialized to define the players. As User is related to BoundingRectangle, the player could know “where can he/she reach?”. Moreover, as in this game a player can be virtually located in several places at the same time, this information could be automatically managed thanks to the relationship between User and VirtualLocation. In this game, players can perform several KindsOfActions (e.g., move, attack, repair, create, select, etc.), being an example of a concrete Action to attack the enemy’s base with the selected troops, thus modifying its State. Moreover, depending on the selected mode of game being played, players would have to achieve different Goals, for instance, to defeat all the enemies or being the first who build a wonder. Therefore, as has been shown throughout the example, most of the design regarding the WA is already provided by the pattern so that the designer only has to customize those elements he/she deems appropriate by inheriting from them.

In order to clarify the relationship between the WA pattern and the WA questions proposed by Gutwin [4], Table I. depicts how the different pattern classes, methods or associations answer the questions.

Finally, it should be noted that, when the system under development does not need all the WA elements, it does not have to implement the whole pattern. For instance, if there is no need to deal with users locations, the Locations class, subclasses and the association relationship with User, should not be implemented.

### IV. PROOF OF CONCEPT

In this section, the implementation of the previous pattern that has been developed by using C# 5.0 [16] is presented as proof of concept. With this aim, Visual Studio 2012 was used to perform such implementation which resulted in the class diagram shown in. Moreover, all the sub-classes for View, Location and Position have been implemented so that the resulting Visual Studio project, which can be downloaded from [17], can be used and/or extended in any CSCW application.

In this implementation, we took advantage of the advanced development functionality provided by C#. Actually, the association relationships among the pattern classes have been implemented by means of C# properties. They are class members that provide a flexible mechanism to read, write, or compute the value of a private field.

![Fig. 2. Radar View of Age of Empires Online](image-url)
It is worth noting that the use of the pattern not only provides designers with guidance in the development of applications with WA support but also with automatic support for its exploitation. Currently, a web service is automatically generated that offers answers to all the questions described in Table I. This means that designers could have support the WA in an automatic way. Moreover, the available implementation of the pattern offers additional functionality. For instance, as aforementioned, this pattern differentiates two different types of past. Therefore, it was decided to implement the recent past by using C# lists and the distant past by using a database. Concretely, a session management system is used so that when a user logs out, all its recent information, corresponding to its last session, is saved in a SQL Server 2012 database [20], whose structure is depicted in Fig. 4. This information available through the distant past querying methods (e.g. whatDid(), whereWas(), howDidHappen(), etc), which are in charge of accessing to the database. They have been implemented by using the Language-Integrated Query (LINQ) [21], which makes code easier to write and understand.

Moreover, the available implementation provides several classes for the management of the different system entities. Concretely, the classes UsersManagement, ArtifactsManagement, Goals-Management and KindsOfActionsManagement were developed in order to offer the mechanisms necessary for creating and accessing to users, artifacts, goals and kinds of actions respectively, thus avoiding “the blob” antipattern (see previous section).

Finally, it is worth noting that, despite the pattern has been implemented by using C#, Visual Studio and SQL Server it could be done by using any other technology such as Java, Eclipse and MySQL [22], which actually is one of our ongoing works.

V. CONCLUSIONS AND FURTHER WORK

As Beck et al. state [23], reuse in software engineering improves the development efficiency, reduce the costs and improve the final software product quality. On the basis of reuse, a design pattern provides a well-known solution to a recurrent problem when designing software, thus reducing time and avoiding errors. Hence, the use of patterns can be applied to the development of CSCW systems, where it is usual to find common entities and behaviors (i.e. several users collaborating on a task to achieve a common goal).

Another common characteristic of CSCW systems is the users’ need of awareness about the other users with whom they
collaborate. Regarding awareness, several interpretations can be found in the literature, being Gutwin’s Workspace Awareness the most widely-accepted one with respect to groupware systems. Thus, in order to answer the questions that Gutwin and Greenberg described in their framework, we presented in this work an object-oriented design pattern consisting of several classes (with the corresponding inheritance and association relationships among them) that provide the answers to these questions. Hence, the application of this pattern in the design of a CSCW system will provide it with a basic structure to answer the WA questions by means of the methods and association relationships of several classes. In addition, that class structure includes the entities commonly used in a collaborative system: users that satisfy their goals by performing collaborative actions over artifacts whose state is modified. Therefore, it can be used as the starting point for the development of complex CSCW systems.

Furthermore, as a proof of concept, this pattern has been implemented by using C# and Visual Studio 2012. It must be highlighted that the advantage of using this pattern is twofold. First, it provides designers with guidance to support WA features in their application. Second, it provides automatic support for its exploitation as a web service is automatically generated that offers answers to all the questions described in Table I. In addition, since the described pattern distinguishes between two kinds of past (recent and distant), the available implementation makes use of a database engine (SQL Server 2012) to manage the information related to the distant past. The information regarding the recent past, which is more frequently accessed, is managed by means of C# lists and automatically inserted into the database when a user logs out. This information can then be accessed by means of the methods parameterized by two dates (from and till) available in the corresponding classes.

Several issues constitute our on-going work related to this pattern. One of them focuses on the development of wizards that guide the designers in the use of the pattern. We are also working on the extension of the pattern to include new elements on the basis of the analysis of the different awareness approaches found in the literature, such as Situation Awareness [24].

ACKNOWLEDGMENTS

This work has been partially supported by the insPIre grant (TIN2012-34003) and the FPU scholarship (AP2010-0259) from the Spanish Government.

REFERENCES