The Absent Participant

More patterns for group awareness

Till Schümmer
FernUniversität in Hagen
Department for Mathematics and Computer Science
58084 Hagen, Germany
till.schuemmer@fernuni-hagen.de

Stephan Lukosch
FernUniversität in Hagen
Department for Mathematics and Computer Science
58084 Hagen, Germany
stephan.lukosch@fernuni-hagen.de

ABSTRACT

The awareness of group members is one crucial aspect in computer-mediated interaction. This paper presents a set of patterns that support groups in which one or more group members are currently away from the group. The patterns help the group to stay aware of the missing user and the user to keep up to date of what happens in the group. The patterns are part of a larger pattern collection for computer-mediated interaction.

Categories and Subject Descriptors

H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—*Asynchronous interaction, Computer-supported cooperative work*

General Terms

Design, Human Factors

Keywords

Design patterns, awareness, groupware

1. INTRODUCTION

In a networked society, new forms of work have evolved over the last two decades. People collaborate over distance in virtual teams. They create virtual collaboration spaces [24] at which they meet at the same or different points in time. Understanding and better supporting distributed collaboration has been the goal of the research area of CSCW (Computer Supported Collaborative Work) [14]. Compared to its early beginnings in the 1980ies, we can currently state that this research area has matured. However, only a few of its findings have entered the main stream of software development. Therefore, we argue that the whole field of computer-mediated interaction should be supported by a pattern language that will help software developers of various domain to better consider the collaboration aspect in their applications. Up to now, we have collected over one hundred patterns in this field [25]. In this paper we present patterns for asynchronous interaction, partially from the collection at `http://www.cmi-patterns.org` and partially up to now unpublished, that have been discussed in a PLoP'06 writer's workshop.

Since we cannot assume that all members of a collaboration space will be present at the same time, it is important that the environment keeps the members informed about activities that occurred during their absence. A concrete example is a virtual team of software developers who are distributed around the world. Rashid from India starts to work on the project at 9:30 local time. At this time, Paweł from Poland is still in bed and Josè from Mexico enjoys a dinner with his spouse. This means that much of the collaboration takes place asynchronously. Coordination of group activities and exchange of information becomes much more difficult. Although all three developers are nominal group members, it is hard for them to stay aware of their activities.

Group awareness has been an important research area in the field of CSCW (Computer Supported Collaborative Work). Dourish and Bellotti [5] defined group awareness as *"an understanding of the activities of others which provides a context for your own activity"*. Providing awareness helps to coordinate the group members' own activities. In synchronous contexts, examples of awareness widgets are Presence Indicators or User Lists that help to detect the presence of a collaborator or Telepointers and Remote Viewports that indicate at which part of a shared document remote users currently put their focus [7], [22], [15]. For asynchronous collaboration, there are fewer examples in the research literature, however, some practices have proven to work.

2. A COMMON STORY

We will explain the patterns in the context of a common story. Imagine that your goal is to support teams of distributed software developers and software customers in the development of the next generation game engine. The members are distributed as shown in figure 1. One team of developers is located in the Rio, one in London and a third in India. The main customer is a large game manufacturer located in Germany who has the goal of building an educational game that helps to better understand water supply in African countries. The game manufacturer has a group of

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PLoP'06 October 21-23, 2006, Portland, OR, USA
Copyright 2006. Copyright is held by the author/owner ACM 978-1-60558-151-4/06/10 ...$5.00.

African pilot users located in Ethiopia and in Malawi.

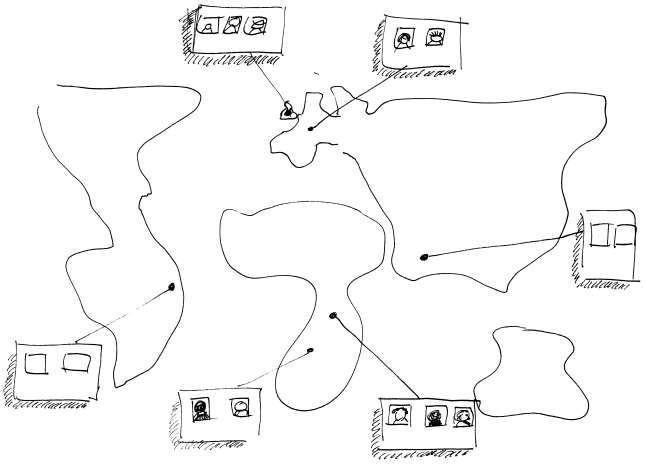

Figure 1: Distribution of team members in the example setting

Different interaction constellations can be found in this hypothetic project: The developers from the London continue to work on the results that were created only some hours before in India. Both software teams communicate to plan the internal architecture of the game engine. In other meetings, the London team collaborates with the German customer who integrates the game engine in his project. The German customer also communicates with some of the developers in India or Rio if time shifts allow an interaction. Finally, the German game manufacturer interacts with his pilot users and collects suggestions from them on how the game could be improved.

Since all teams are distributed and potentially work at different points in time (remember the time shifts), the awareness of other team members becomes a crucial aspect for making interaction and coordination between the individual participants possible.

3. A PATTERN LANGUAGE FOR ABSENT PARTICIPANTS

We can approach the problem of group awareness from different perspectives. Assuming that one user leaves the collaboration context for a longer period of time, it is important that (a) the group becomes aware of the other user's absence so that they can adapt their group process if needed and (b) the absent user is aware of important changes in the collaboration space so that he can more easily resume collaboration after his absence. This is reflected by the patterns in this paper.

Figure 2 provides an overview on the patterns. Patterns are shown as rounded rectangles. Numbers refer to the section in Chapter 3 where the pattern can be found. The arrows between the patterns denote a reference between the patterns. If a pattern has been considered for application, you should also consider all patterns that are related from the original pattern. Bordered areas cluster the patterns with respect top their main focus.

When implementing a system that supports absent participants, you should first ensure that activities of other users are logged and therefore consider patterns from the *Logging* cluster. The Activity Log and the Activity Monitoring pattern support this. Since these two patterns were published before [22], we only include their thumbnail in this paper:

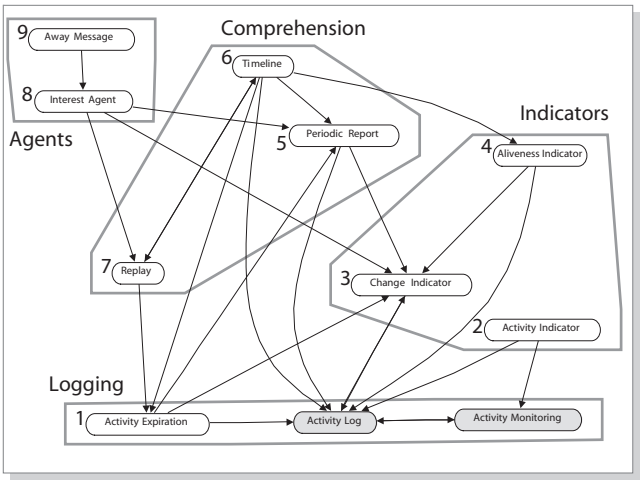

Figure 2: Patterns for asynchronous group awareness described in this paper

Activity Log

Problem: Merging two users' (past or current) activities is a difficult task. It requires the activities to be transferred to the same context and the goals aligned. Many applications do not provide access to the history of an artifact, its use, and its evolution, however. Thus merging is vulnerable to errors and often collaboration does not take place, since the effort of merging exceeds the expected gains from a collaboration.

Solution: Remember all activities – not only modifications, but also read accesses – that users perform on shared artifacts in a log. Provide access to the log so that it is possible to understand (and merge) the various users' activities.

Activity Monitoring

Problem: Many proprietary tools are not designed for extendability. They do not provide means to modify the application's behavior. This makes it difficult to add automatic tracking of user's activities, which you would need to provide awareness.

Solution: Add an additional layer in the communication between the application and the shared data to monitor the user's activities. Allow other parts of your application (e.g. a Activity Log or a User List) to subscribe to monitored activities.

These two patterns help to capture events. In line with [9], we define an event as *"any significant change in the state of an observed object"*. The Activity Expiration$_{\rightarrow 3.1}$ pattern which is added to the *Logging* cluster in this paper argues for removing events from the activity log if it grows too large to avoid information overload.

The log data can now be analyzed and visualized in different ways. The cluster labeled *Indicators* includes patterns that help to better understand traces of users in the system. The Activity Indicator$_{\rightarrow 3.2}$ shows where the other users have been active while the Change Indicator$_{\rightarrow 3.3}$ vi-

sualizes modifications to artifacts that the absent user has not yet seen. The ALIVENESS INDICATOR$_{\rightarrow 3.4}$ finally shows whether or not a user has been active at all.

These patterns focus on individual activities. After a longer period of absence, this is not sufficient to get a holistic idea of the group's progress. The three patterns in the *Comprehension* cluster therefore support the user in better understanding sequences of activities. The REPLAY$_{\rightarrow 3.7}$ pattern replays activities like in a movie. The TIMELINE$_{\rightarrow 3.6}$ provides a quick overview by chronologically visualizing changes. The PERIODIC REPORT$_{\rightarrow 3.5}$ avoids a large divergence between the absent user's knowledge and the group progress by frequently informing the absent user on changes.

The last cluster in Figure 2 includes two patterns that show how the user can stay "present" without "being there". The idea is that an *Agent* takes the role of the user while the user is absent. The INTEREST AGENT$_{\rightarrow 3.8}$ collects activities that are relevant for the absent user and the AWAY MESSAGE$_{\rightarrow 3.9}$ communicates the absence to the group whenever the absent user is required for interaction.

As mentioned in the introduction, this paper only presents a small part of a larger pattern language on computer-mediated interaction [25]. Whenever patterns reference other patterns of this language, you can find short versions of these patterns on `http://www.cmi-patterns.org` or full versions of the pattern in [25]. Note that after the PLoP conference, most of these patterns evolved so that they could be included in the book on patterns for computer-mediated interaction [25]. This shows the impact that a writer's workshop can have: through fruitful discussions, the topics presented in the patterns became more focussed, the pattern's inherent forces became clear, the coherence of the pattern language increased, and the concepts presented in the patterns were be related to experiences made by professionals participating in the workshop.

For the patterns present in [25], we decided to include an abbreviated version in this paper. These are ACTIVITY INDICATOR$_{\rightarrow 3.2}$, CHANGE INDICATOR$_{\rightarrow 3.3}$, ALIVENESS INDICATOR$_{\rightarrow 3.4}$, REPLAY$_{\rightarrow 3.7}$, TIMELINE$_{\rightarrow 3.6}$, PERIODIC REPORT$_{\rightarrow 3.5}$ and AWAY MESSAGE$_{\rightarrow 3.9}$. You may consider these short versions of the pattern as an appetizer that should be sufficient to understand how the pattern works. To the same extent as we explain the problem and the solution, we include danger spots, namely potential negative effects when applying the patterns. They are mainly the result of the discussion at the writer's workshop: participants shared their observations with patterns like AWAY MESSAGE$_{\rightarrow 3.9}$ and complained about information flooding experienced by too extensive awareness mechanisms. We consider this as an important aspect of awareness patterns in general. How to balance the need of staying aware with the desire of spending low efforts on information processing is a challenge that needs to be addressed when applying the patterns in a specific context. To our experience, the discussion of danger spots makes it easy for the designers and the users to find a good balance.

More detailed information like discussions of the pattern's rationale, extended examples, or hints for applying the pattern in practice are provided in the book. For space reasons, two patterns could not become part of the computer-mediated interaction book and are only presented in this paper: ACTIVITY EXPIRATION$_{\rightarrow 3.1}$ and INTEREST AGENT$_{\rightarrow 3.8}$. They are included in this paper in full length.

3.1 ACTIVITY EXPIRATION

Context: You are providing asynchronous awareness like information on who has edited a specific page.

Problem: Awareness information helps to make a user understand what other group members have done in a collaboration space. They show other user's traces and therefore make the collaboration space a living place. However, these traces become too confusing if they stay in the space forever.

Scenario: Susan has been very active in the past and thereby gained a top position in the community's HALL OF FAME. However, she stopped her participation half a year ago and is no longer responding to requests. Claire who is seeking for assistance is thus mislead by the information found in the HALL OF FAME that suggested her to contact Susan.

Symptoms: *Consider to apply the pattern when ...*

- users complain that they are informed about old activities that are not relevant anymore.
- users react to an activity but the other user who performed the activity is no longer aware of what she did in the past.

Solution: **Mark activities as outdated after a specific point in time or after a user has noticed the activity. This means that these activities are no longer considered when providing awareness information for the user.**

Collaborations: Activities are stored in an ACTIVITY LOG and used to provide awareness on past actions. The pattern proposes to have two triggers for marking activities as outdated:

1. A user can explicitly request that an activity is marked as outdated. This means that a flag is set in the activity object.
2. The system periodically checks activities and flags those activities that are too old.

In the first case, the flag has to be an attribute that is only associated to the user that has set the flag. This is necessary because different users can request that an activity is outdated at different points in time. In the second case, a global flag is set that is valid for all users.

When the system calculates awareness information, it only considers those activity records that are not marked as outdated.

Rationale: When providing awareness information, it is always important to balance the amount of information so that it is not too complex for the user. Otherwise, the user will be distracted by the awareness information. The question of what to filter in the process of providing group awareness has to be stated in each design of an awareness widget.

One way to filter awareness information is to assume that a user will remember information that was presented to her in the past. If a user, e.g., has been informed about new versions of a document in a PERIODIC REPORT$_{\rightarrow 3.5}$, one can assume that she will have processed this information.

Telling her about the new document on the next day again will probably rather distract the user. The same is true for activities that do no longer have an impact on the group interaction. If a user, e.g., created a document to plan the next days of collaboration, it will often make little sense to inform another user who has been absent for a year about this planning document a year later.

It is therefore important that old traces are no longer visible. The decision when it is appropriate to hide old activities is often user dependent. A user who has seen the newest version of a specific part of the collaboration space will show less interest in the activities for that part as a user who has not yet found the time to review the changed artifacts.

One could argue that the activities could also be deleted. However, in some cases the activities can become important again. One example is that users do no longer remember who collaborated on an artifacts in its early days. Being able to find these people can be important if implicit knowledge is requested from them.

Check: *When applying this pattern, you should answer these questions:*

- What kind of automatic triggers will you use to outdate activities?
- Will you allow a user to outdate activities manually, e.g., by providing a command that outdates all activities for a specific user?
- Can you outdate activities after you presented them to the user?

Danger Spots: In cases where the activities are stored in a database, the storage becomes more complex if the activities are outdated for the individual users. It can then make sense to only remember the time of the oldest activity a user is interested in. Users can in this case decide to forget all activities that are older than a specific age.

Known Uses:

BSCW [1] maintains an Activity Log to visualize modified artifacts using Change Indicators$_{\rightarrow 3.3}$. These indicators however do not vanish unless the local user decides to catch up all changes. This results in the situation where the change warnings are no longer shown.

Motorola E1000 is a mobile phone in which the SMS configuration may be appropriated in order to keep only a specific number of short messages and skip old messages.[1]

Firefox `http://www.mozilla-europe.org/en/products/firefox/` is a web browser that is able to automatically forget all visited pages after a user specified period of time. Links to these pages will then be displayed as unseen again.

Related Patterns:

Activity Log**:** The Activity Log stores all activities that were performed in a collaboration space. The importance to consider not all of these activities is raised by the Activity Expiration pattern.

[1] `http://direct.motorola.com/ENG/web_producthome.asp?Country=GBR&language=ENG&productid=29265`

3.2 Activity Indicator

Context: Users are geographically distributed and interact in a highly synchronous session that involves frequent turn-taking and request-response interaction.

Problem: Users need time to perform a task but only the results are shared among them. In a collocated setting users are accustomed to perceive non-verbal signals such as movement or sounds when another user is active. If the users are distributed, these signals are missing. Users are therefore not aware of other users' activities, which can result in conflicting work or unnecessary delays.

Scenario: Rick and Dimitri both decided to work on graphical aspects of the game engine project. Rick started to work on the rendering of places, while Dimitri looked at the rendering of actors. At one point Dimitri started changing some code that he had currently checked out to his workspace. He planned to check the code in again after he had tested his modifications. However, if Rick also wanted to modify the same section of code, both user's changes would need to be merged after they had finished their tasks.

Symptoms: *Consider to apply the pattern when ...*

- Users dislike distributed interaction because they do not know what the other users are doing.
- Users perform concurrent actions.
- Users wait a long time for another user's action, even if the other user does not act at all.
- Users act at the same time but do not necessarily share the same focus.
- Users do not want to be distracted from their current task, but still feel the need to stay aware of other users.

Solution: **Indicate other user's current activities in the user interface. To reduce interruptions, use a peripheral place or a visually unobtrusive indicator.**

Collaborations: Provide a user interface element in a peripheral location that shows whether remote users are active and what they are currently doing. The current activity of a user can for example be determined using an Activity Log. Ensure that remote users' activities are shown in the user interface immediately the start to act. Hide the activity if no more activities are detected from the remote users (for example, there is no keyboard input for a specific period of time).

Danger Spots: If many users collaborate, displaying their various activities becomes difficult. Therefore, reduce the information provided and cluster the information when several users are performing the same activity, for example.

Known Uses:

MSN Messenger shows when another user is typing a message in the status bar of the chat window. This helps the local user to better judge whether a reply can be expected.

Palantír [20] is a group awareness component that extends a software configuration management system. Palantír tracks the activities of project team members to provide information about their actions.

Mail Clients like Thunderbird `http://www.mozilla.com/thunderbird/` provide a status icon whenever a new message arrives.

Related Patterns:

Attention Screen can be used to filter the awareness information shown in the Activity Indicator.

Activity Log describes how to store information on the users' activities and can be used to determine a user's current activity.

3.3 Change Indicator

Context: Users work on independent copies of shared artifacts.

Problem: While users works on independent local copies of artifacts, their checkout frequency for the artifacts may be low. As a result, they may work on old copies, which leads to potentially conflicting parallel changes. The conflict is worse if two parallel modifications have contradictory intentions.

Scenario: Marc has made some major improvements to the graphics engine. The most important was that he changed the coordinate system from cartesian to polar. Marc documented this change in the manual of the game engine. Martin, who uses the game engine, was not aware of this change. He thinks that he knows how to use the game engine and does not often read the manual. He therefore uses the graphics engine part with cartesian coordinates and is confused when the images look very strange.

Symptoms: *Consider to apply the pattern when ...*

- Users apply changes to artifacts based on outdated knowledge of the artifact's state.
- Users report that they would have done things differently if they had been aware of the current state of the artifact.
- Users frequently change artifacts.

Solution: **Indicate whenever an artifact has been changed by an actor other than the local user. Show this information whenever the artifact or a reference to the artifact is shown on the screen. The information should contain details about the type of change and provide access to the new version of the artifact.**

Collaborations: Figure 3 shows how the different participants collaborate. Note that we have chosen to demonstrate the pattern on the assumption that shared artifacts are documents kept on a central document server. Without limiting generality, this can be transferred to any shared artifact that is kept at a specific place and that has to be copied to a local system before it can be changed.

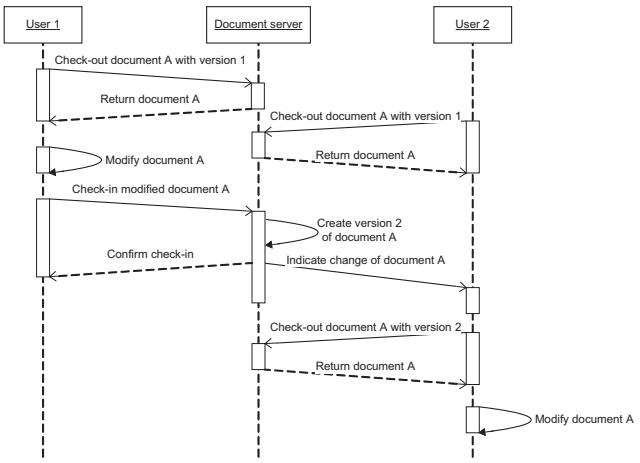

Figure 3: Two users are working on a shared document

In Figure 3 user 1 downloads a document A from the shared document server. This document has the version 1. Later on, user 2 also checks the document A. Both users now have independent copies of the document A which have the same version. User 1 modifies the local version of the document A and transfer this modified version to the shared document server. The server creates a new version of document A and informs user 2 about the new version. User 2 has not modified the local copy of document A, however, so checks out the new version before performing the planned modifications.

Danger Spots: Even when indicating changes, people might ignore the indications. To overcome this problem, it is necessary to establish a social protocol that defines how Change Indications should be handled.

Changes can be complex, and providing details about them can be complex too. In that case, consider providing a comparison view that contrasts a local user's state of a changed artifact with its state as seen by a remote user.

If changes occur too frequently, most artifacts will be shown as changed artifacts, resulting in constant searching for changes and insufficient time to perform constructive actions. One solution could be to highlight unchanged artifacts explicitly, so that users can be confident that such artifacts need no further attention.

Known Uses:

TUKAN. The programming environment TUKAN [21] uses a weather metaphor to display change warnings. A bold lightning symbol tells programmers that a specific artifact has been changed.

WinEdt. The text editor WinEdt (`http://www.winedt.com/`) buffers the current file in memory while the user performs edit operations on the file – as do many other editors. If someone or something else has changed the file, WinEdt displays a warning to indicate that the current file has been modified outside of the application.

Related Patterns:

ACTIVE NEIGHBORS should be used if artifacts are semantically related. In this case, it is important to highlight not only changes to the current artifact, but also changes that might have an impact on the current artifact at a semantic level.

ACTIVITY LOG Use an ACTIVITY LOG to store the activities that are used to calculate conflicting activities.

USER LIST When attached to artifacts, a USER LIST is comparable to a CHANGE INDICATOR in that it also displays activities on artifacts. The main difference is that a USER LIST only consider activities that are still active. In most cases, CHANGE INDICATORS show activities that are completed.

3.4 ALIVENESS INDICATOR

Context: Users collaborate asynchronously on shared artifacts or in shared virtual or non-virtual collaboration spaces.

Problem: Users who work mainly asynchronously only experience a small subset of activities that take place in the collaboration space. Specifically, they cannot easily see whether other users have been active during their absence. This makes it hard to experience life in the group.

Scenario: Marc just returned from a business trip. He enters his office and wants to continue with the work on the graphics component that he started last week with Carla. However, he does not know whether or not Carla has also done any work in this area while he was out of office.

Symptoms: *Consider to apply the pattern when ...*

- Users complain that others have stopped participating, although the silent users still follow the interaction.
- Users ask the whole group whether or not they are still participating.
- Users ask each other to visit the collaboration space but are not sure whether or not other users have followed their request.

Solution: **Show an ALIVENESS INDICATOR with the user's virtual representation. For users that have performed activities in the collaboration space recently, use a picture for their indicator that looks very *alive*. Use gradually less *lively* pictures to represent periods of inactivity. Make the picture look like something for which the user can take responsibility.**

Collaborations: The system keeps track of users' last activities in the collaboration space. When visualizing the members of the collaboration space (in the context of the collaboration space), the system calculates the time span since the user's last activity. Depending on this interval, the system selects a different indicator to show with each user's representation. For short time spans and hence recent activity, the indicators symbolize a high degree of vivacity, while longer time spans are symbolized by less lively pictures.

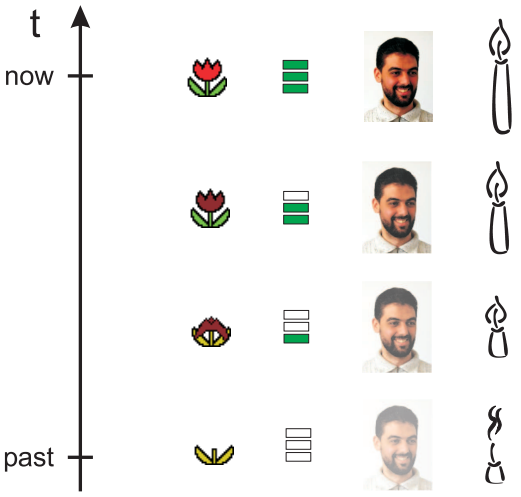

Figure 4: Aliveness Indicators

Figure 4 provides examples of aliveness indicators: a withering flower, a declining bar chart, a fading picture, or a candle burnt down.

Optionally, the indicator can provide additional information, for example by using a tooltip. It could show the time when the user was last active, or provide details about what the user did when active. The indicator can also have different scopes. For a shared workspace system, there can be one indicator for each user in each workspace. In this case, it makes sense to use an indicator that represents an artifact that the user must take care of, such as the flower in Figure 4. In systems where the collaboration space is less important, there can be one global scope. This means that the user has the same indicator for the whole system. The fading user image is an example of such a visualization.

The second decision for the scope is whether the indicator should be bound to an individual or to an artifact or region in the collaboration space. An example of the latter could be that the place is symbolized by a virtual flower and that every activity in the place "waters" the flower.

Danger Spots: The motivation to keep an activity indicator alive is probably the most important pitfall in this pattern: users can just pretend participation. Whether or not their participation is relevant to the group is not measured by the pattern.

An interesting analogy to this was reported by Dave West at the PLoP writer's workshop: he remembered a UNIX system that scheduled interactive processes with higher priority. The reason for that was that live processes should respond quickly. However, this handicapped non-interactive processes: what happened in his lab was that users started to press the space bar repeatedly just to pretend that their processes were interactive!

In the case of the ALIVENESS INDICATOR, the same phenomenon may emerge if the number of activities is taken as the only measure for participation. You could consider grouping different activities into different classes, so that some activities have a higher impact on the ALIVENESS INDICATOR than others. However, this may only shift the problem somewhere else: users could find new ways to trick the indicator.

In general, this relates to the issue of trust. Aliveness indicators can create a large social pressure to participate, especially if managers observe them. In contexts in which participation is less mandatory, you should think about using Masquerade to allow users to turn their indicator off. In any case, this should be coupled with the Reciprocity pattern, so that users who turn their indicator off cannot see other users' indicators.

Be aware of different cultural perception for the icons you use for Aliveness Indicators. In some cultures, an extinguished candle can be interpreted as a sign of death, so people might relate it to a feeling of being dead for the community. Icons thus need to be carefully chosen to suit a community, and should be more abstract than concrete if there is cultural diversity in the community. The same is true for the use of colors: in some cultures, white is for example a color of freshness and birth, while other cultures connect associate it with mourning and death.

Using too many indicators, such as at every place in the collaboration space, can lead to a situation in which users constantly chase their indicators to keep them alive. Consider for example the example of a virtual flower that is bound to a workspace. If users are allowed to enter many workspaces, they will also have many flowers and will need to look for them even if the workspace is deserted for a longer time. A solution to this problem could be either to reduce the number of places that have flowers for specific users, or to slow down the aging of the flowers so that they only fade when both other users are active and the owner of the flower is absent. This however complicates the calculation of the flower's age.

Known Uses:

Flowers in CURE. In the CURE web-based collaboration space [23] flowers are used to indicate when a user is active in a group's space.

XING (`https://www.xing.com`) is a social networking system that shows an *activity meter* on each user's contact page that represents how active the user has been recently.

Related Patterns:

Change Indicator$_{\rightarrow 3.3}$ shows a modification on the relevant artifact. The difference between this and an Aliveness Indicator is that only modification activities are shown, and that time in most cases does not play an important role in the Change Indicator pattern.

Activity Log allows collection of activity information about users and calculation of how the Aliveness Indicator should be displayed.

User List helps to understand the presence of a user better. An Aliveness Indicator is also applicable in semi-synchronous or asynchronous settings.

User Gallery provides a place to display an Aliveness Indicator.

Away Message$_{\rightarrow 3.9}$ Users who plan an absence can set up an Away Message to explain to the community why their Aliveness Indicator indicates inactivity.

Virtual Me can be combined with an Aliveness Indicator. Whenever other users browse the page describing the user, they can also see how "alive" the user is.

Reward is another way of honoring participation, but this time by considering the quality of other users' actions. When users perform valuable actions, they receive an award. The difference to an Aliveness Indicator is that rewards normally do not fade.

3.5 Periodic Report

Context: Users collaborate asynchronously by modifying shared objects.

Problem: Changes in indirect collaboration are only visible by inspecting a changed artifact. Users want to react to actions on artifacts, but they cannot predict when these actions will take place.

Scenario: In January Weigang filed a bug report on a security problem in the login mechanism of the game engine. He then checked the affected components frequently for an update. In January he did this daily, since he really needed the security fix. Nothing happened, however, so Weigang reduced the frequency of his update checks. Now, four months later, he only scans the files every fortnight and has given up hope of a fix. Reflecting on the last few months, Weigang regrets that he spent so much time looking for updates.

Symptoms: *Consider to apply the pattern when ...*

- Users rely on each others' activities but cannot predict when they will take place.
- Users frequently scan for changes but rarely find any.
- Collaboration takes longer than it should because users do not scan for changes as frequently as they appear.
- The community performs many modifications a day, so that direct notifications of each change would consume too much attention or would be too expensive.

Solution: **Inform users periodically about the changes that took place between the time of the current report and the previous one.**

Collaborations: Users defines an interest profile manually or automatically based on their access rights in the collaborative system. They also define a notification interval and a communication channel by which they would like to be notified.

After the interest interval has passed (in most cases at night), the system checks whether artifacts matching any interest profiles have been modified within the last time interval. If this is the case, the system puts meta-information on the change into a periodic report. The periodic report, with information on all matching modified objects, is sent to the users using the requested communication channel.

Meta-information can for example contain a short description of the artifact, information about the person who modified it, and the time and type of the modification. It should include a quick reference to the changed artifact to ease access to it.

The check for changed artifacts can take place in two alternative ways:

1. The system can query the timestamps of all objects and search for those that fall within the notification interval. This has the advantage that the artifacts only have to carry a timestamp and no additional data structures are needed to track changes. Information on the performer of the change needs to be stored with the artifact if such information is required in the periodic report.

2. The system tracks all changes in an ACTIVITY LOG and queries the ACTIVITY LOG for activities that took place during the notification interval. Since the activities carry all the required meta-information (performer, time stamp, and the type of activity), this information does not need to be part of the artifact. However, the number of recorded activities may soon grow and slow the system down.

Danger Spots: The report can be considered as spam. Make sure that the users know how to tailor the report to their needs.

In most cases it is advisable to avoid empty reports. However, subscribers could think that their report had been lost if they didn't receive one.

Make sure that the report is structured in a way that can be easily grasped. Provide enough information on the artifacts to allow a reader to filter irrelevant changes without looking at the specific artifact in the system.

Known Uses:

BSCW is probably one of the best-known collaborative systems to make use of PERIODIC REPORTS.

Forums like Yahoo Groups (`http://groups.yahoo.com`) often provide options for controlling the frequency of messages sent to subscribers.

CURE sends a daily report every night. The report lists changes that occurred in rooms to which the recipient has access.

e-Commerce websites often allow customers to store their interests and their e-mail address on the server. Whenever a new item is added or an old item is changed, the site sends a notification e-mail informing the potential customer that there might be something new of interest on the site.

Related Patterns:

ATTENTION SCREEN An attention screen filters notifications and contact requests to ensure users' privacy. It can be combined with a PERIODIC REPORT to ensure that users stay informed about the activities in a collaborative environment. It can also enhance acceptance of the PERIODIC REPORT, since it allows users to define the information that should reach them via the periodic report.

CHANGE INDICATOR$_{\rightarrow 3.3}$ A change indicator provides information about changed artifacts in the same context as the artifact itself. The notification that an artifact has changed is attached to the artifact. In contrast, a PERIODIC REPORT externalizes this information and transmits it to users' work contexts outside the system, for example via users' mailboxes.

ACTIVITY LOG An ACTIVITY LOG keeps track of all activities in a system. A PERIODIC REPORT can be generated from the ACTIVITY LOG by querying it for activities that took place on relevant artifacts since the previous report.

3.6 TIMELINE

Context: Your system supports long-term asynchronous and/or synchronous interaction.

Problem: Not all users participate in collaborative sessions continuously. This makes it hard to understand who is working with whom on what topic. Without such an understanding, however, users lack the orientation and coordination required for group interaction.

Scenario: Maurice tries to understand what Paul and his colleagues implemented last week while Maurice was on vacation. He knows that the other developers discussed the strategies for their work frequently and then split work between themselves. He decides to examine the change notifications that were circulated by e-mail, but even after this he still lacks a holistic picture of the development activities.

Symptoms: *Consider to apply the pattern when ...*

- Users complain that others do not participate, although they do participate.

- Users stop participating, but this is not detected by the group.

Solution: **Display the activities that took place in a workspace as a timeline.**

Collaborations: The timeline is a two-dimensional diagram that relates the time of an activity with either the artifact used in the activity or the performer of the activity.

First group the activities monitored in the ACTIVITY LOG by the days on which each activity took place. Then show, for each day, the activities that took place on that day. Separate the days using bars.

Display each activity as an icon or a dot in the diagram. Use different icons for different users when showing the artifacts accessed by the activities on one of the axes. When showing users on one of the axes, think about different icons for the different artifacts that were accessed.

Use dynamic data visualization techniques such as DATATIPS or LOCAL ZOOMING [26] to support the display of long activity logs with many artifacts. This means that additional information is provided on request. Connect the display of activities with the documents that were accessed, so that the timeline can be used for navigating to shared objects.

Danger Spots: Ensure that you select the data set with the higher cardinality for the Y axis. If your group, for example, has five members who work on fifty documents, the documents should be shown on the Y axis and the members should be shown using different colors or icons. This leads to a diagram with fifty lines and five different icons. Otherwise, one would have a diagram with just five lines but fifty different icons, which would probably be harder to read.

One problem with this pattern can be scalability. As Ganoe at al. [12] evaluated in a field study, a timeline can become "less effective (and even cluttered) if there are frequent changes to all the documents."

Known Uses:

Virtual School is a collaboration space for student interaction. In a user study [2], the authors of the Virtual School environment, found several breakdowns in collaboration that had their causes in a lack of activity awareness. One solution was to integrate a timeline into the students' workspace. The resulting system was then called the BRIDGE awareness center [11].

Babble Timeline [6] is a visualization widget for better understanding the history of chat conversation. It shows up to one week of the chat log recorded in a Babble chat.

CVS History Explorer `http://www.eclipseplugincentral.com/Web_Links-index-req-viewlink-cid-819.html` is a tool that visualizes the history of files stored in CVS, a versioned Shared File Repository. Users can select an artifact and open the timeline for the artifact, which is based on the Flow history visualization first proposed by [27].

Related Patterns:

Replay$_{\rightarrow 3.7}$ also addresses the problem of explaining the activities that took place in the collaboration space to an absent user. The difference is that a Timeline visualizes activity information, while the Replay pattern shows which artifacts the activities changed.

Periodic Report$_{\rightarrow 3.5}$ A Periodic Report provides a more detailed view of changes in a collaboration space. It is well suited to short time spans, but will become very complex when it shows a longer period. A Timeline abstracts further from the activities and is therefore capable of providing a longer overview of activities.

Activity Log A Timeline displays the activities stored in the Activity Log.

Aliveness Indicator$_{\rightarrow 3.4}$ helps to detect the fact that a user has stopped participating. This can be seen in the Timeline when there are no more entries for a specific user – especially when the timeline display shows users on the Y axis.

Immutable Versions For each activity, you should be able to link to the version of the document that resulted from the activity. This requires that you keep all versions of the shared artifact, as described in the Immutable Versions pattern.

Timeline in the context of project retrospectives [17]. The technique of project timelines has been widely used in the context of project retrospectives, and captured as patterns by various authors, e.g. [13]. The basic idea is that members of a project team place notable project events on a visual timeline. The technical Timeline presented in this pattern can support the creation of a project timeline. When using a technically generated Timeline in a retrospective, you should allow users to attach Shared Annotations to the Timeline so that they can comment on the events.

Dynamic Queries **and** Data Brushing [26] discuss how complex diagrams like a Timeline can be explored interactively. In the Dynamic Queries pattern, users can control a set of filters that defines which data elements are included in the diagram. In the context of the Timeline a filter could reduce the set of users who are included in the diagram. The Data Brushing pattern suggests simultaneously showing different diagrams and allowing the user to select data in one of the diagrams, with the effect that the other diagram shows more detailed information based on the data selected in the first diagram. Translated to the context of a Timeline, you could consider having one global diagram for the collaboration space, with detail diagrams that are shown after the user selects a set of documents in the global diagram.

3.7 Replay

Context: You allow users to join, leave, and rejoin a collaboration at different points in time.

Problem: When users join an ongoing collaboration as latecomers or when users rejoin a collaboration after a time of absence, it is hard for them to understand how the current state of the collaboration has been reached, or what has changed since their last participation, by only perceiving the current state of the collaboration.

Scenario: Paul and Susan performed an extremely successful pair-programming session. The next day, Susan wants to continue the session but Paul has contracted flu and cannot participate. Susan therefore asks Liam. Unfortunately, Liam does not know what has been going on in Paul's and Susan's session, which make it hard for him to start collaborating with Susan.

Symptoms: *Consider to apply the pattern when ...*

- Users have problems participating in a collaboration from its beginning.
- Users have problems understanding how the current state of a collaboration has been reached.

Solution: **Capture all changes to the shared objects used in the collaboration in an** Activity Log**. When users join or rejoin a collaboration, replay the captured changes to show them how the current state of the collaboration has been reached.**

Collaborations: To replay the sequence of changes by which the current state of a Collaborative Session has been reached, it is necessary to capture all changes that are applied to the shared state and to store these changes in a log (see Activity Log).

Two cases have to be distinguished: with or without a central server. In the simplest case, a server can be used as a provider for the latecomer. This is for example the case

when using Centralized Objects or Mediated Updates to communicate state changes. In both cases, the server has to be enhanced to keep a log of all state changes already applied to shared state. Depending on how state changes are distributed, this might either be a log of Distributed Commands or a set of Immutable Versions for all shared objects.

The client system joining the collaboration is called a "latecomer". When a central server is available, a latecomer can contact the server as provider and request the log of state changes. After the latecomer has contacted the provider, the provider includes the latecomer in future communication about state changes. From that point on the latecomer buffers all messages about state changes.

After including the latecomer in communication, the provider supplies the latecomer with a log of state changes. As soon as the latecomer has received the log, it starts to re-execute all state changes and display their execution in the user interface. While executing the state changes, the latecomer displays a control panel that allows the speed used to re-execute the state changes to be set. After replaying the log, the latecomer checks to see whether it has received further state changes. If there are state changes in the buffer, the latecomer executes them. When the buffer is empty, the latecomer has a consistent current state, stops buffering state changes, and participates in the session.

If there is no central server, it is not necessarily the case that any of the clients have been in the session from its beginning and so know of all state changes. To solve this issue, you can model the state changes as shared objects again. All clients synchronize their logs by means of the synchronization mechanisms for Replicated Objects and keep a full log of all state changes that can then be sent to latecomers. State Transfer can then be used to provide a latecomer with a copy of the log.

Danger Spots: Transmitting the complete log of state changes might take a lot of time, especially if a latecomer is joining very late in a session. To overcome this issue, make regular copies of the shared state and let latecomers choose the point of time at which replay starts. Then select the copy of the shared state that is closest in time before the time selected by the latecomer. Transmit this copy and only the commands that have been executed afterwards to the latecomer. The latecomer then uses the copy to initialize shared objects and start executing commands from the selected point in time.

It may not be feasible to create a full replay of the interaction. The system can only replay those parts that were performed using the system. However, collaboration normally takes place using many interaction channels. The probability that not all of these channels can be captured and replayed is thus quite high.

A full replay can be too fine-grained, since it shows all operations. The common way to overcome this issue is to abstract from fine-grained changes to high-level changes. You should therefore consider replaying composite changes in one step and augmenting the replay with meta-information. Finding the right level of abstraction is however domain- and case-specific.

Known Uses:

Collaboration Bus [3] is a groupware development environment that offers a service that allows sessions to be replayed.

DreamObjects [18, 19] is a groupware framework that keeps the log as a replicated object. It therefore does not need a central server to act as a provider of the log. When joining, users can choose what percentage of the current should be replayed and the delay between display of state changes.

CatchUp [16] is a plug-in for the Eclipse development environment that allows record and replay of refactorings.

Related Patterns:

Activity Log describes how to log all changes to the shared state.

Centralized Objects allows a central server to be chosen as session state provider for a latecomer.

Collaborative Session allows users to plan and coordinate synchronous collaboration. Latecomers need the current state of the session.

Distributed Command allows state changes to be encapsulated.

Immutable Versions allows different versions of a shared object to be identified.

Mediated Updates allows the mediator to be chosen as state provider for a latecomer.

Replicated Objects can be used to keep a replicated log of state changes.

Decentralized Updates requires that all or a subset of all clients keep the log of state changes, because the participants of a Collaborative Session communicate in a peer-to-peer network, so there is no central server that can be chosen as change log providers for latecomers.

State Transfer directly transfers the current session state to a latecomer.

Timeline$_{\rightarrow 3.6}$ shows the orchestration of different activities by means of a diagram. A Timeline can be compared to a script of activities, while Replay executes or animates the script.

3.8 Interest Agent

Context: Users interact in a long-term topic-based interaction.

Problem: In order to be able to follow and understand long-term interaction, users often have to participate in the evolution of the topic. But for time reasons not all users can participate in the group throughout the whole process.

Scenario: At the kick-off meeting for the new game engine, the main office invited all potential customers to participate in the specification of the engine. Martin flew to London for this purpose and shared their thoughts with Paul and Maurice. Martin would have liked to stay updated on the project's progress since this will make it probably easier to

use the game engine in his next project. But unfortunately, Martin had to return to his office and spend all his time in the other project his company is working on.

Symptoms: *Consider to apply the pattern when ...*

- Interaction that is relevant to many group members takes place with only a small set of group members present.
- Users want to react to group activities but they don't know when such activities will happen.
- Users interact in a general purpose interaction space but are only interested in a specific subset of topics.

Solution: **Allow the users to place an interest agent at the interaction space that keeps track of relevant changes.**

Collaborations: A user participates with other users in an interaction space. When the user temporarily leaves the interaction space (e.g., because he wants to interact with another group), his interest agent takes his place.

The interest agent is a software component that is capable of observing an interaction space. It acts in this space as a representative of the user. Whenever there is an activity in the interaction space observed by the interest agent, the agent informs the corresponding user.

The interest agent should be tailorable so that the user can decide, which activities in the interaction space are of interest for him.

In case of small groups, the interest agent should be shown to the remaining group so that the other group members stay aware that the absent user follows the interaction (User List).

From a technical perspective, the interest agent is an Observer [10] of an event stream. New events trigger the interest agent which then checks whether the event matches one or more of the interest agent's trigger conditions. If this is the case, the interest agent creates a notification for the absent user.

Rationale: Since the interest agent follows the interaction, the owner of the agent can be sure that he will not miss important actions in the interaction space.

Check: *When applying this pattern, you should answer these questions:*

- How can you make the interest agent tailorable?
  - ▷ Are there specific topics that can serve as filters for relevant interaction?
  - ▷ Are there different types of activities that are of different interest to the owner of the interest agent (the creation of an object could, e.g., be of greater importance than the removal of the object).
  - ▷ Does it make sense to follow only specific users' actions?
- How do you inform the owner of the interest agent? Is a daily notification (using a Periodic Report$_{\rightarrow 3.5}$) sufficient or should the interest agent provide immediate feedback?
- Is it important that other users stay aware of the absent user?

Danger Spots:

- Selecting the appropriate level of detail for the report is difficult. The reported information can range from the changes to artifacts that are reported in a Periodic Report$_{\rightarrow 3.5}$ up to communication content or navigation activities that are reported by using Replay$_{\rightarrow 3.7}$. Therefore, users must be able to tailor the reported information to their needs.
- Often important information is captured semantically, e.g. in a audio discussion among team members. This kind of information is difficult to capture and report.

Known Uses:

Netnews: Many news readers allow the user to tag relevant threads. The reader will track changes in this thread and inform the user when new messages are added to the thread.

ELVIN [8] is an event notification infrastructure that allows users to configure interest agents to observe the stream of events in the collaboration space.

Related Patterns:

Periodic Report$_{\rightarrow \mathbf{3.5}}$**:** The notifications of the interest agent can be accumulated in a periodic report. This has the advantage that the owner of the agent will not be disturbed during the day.

Change Indicator$_{\rightarrow \mathbf{3.3}}$ also addresses the problem that a user is unable to perceive all changes immediately. But instead of informing the user when the change occurs, the Change Indicator attributes the visualization of the changed artifact in order to show that this artifact has not been seen by the interested user.

User List covers the aspect of the interest agent pattern that argues to visualize an absent user's agent. However, there is one big difference: The Interest agent shows users who are interested but not there while the User List pattern shows users who are currently following the group process (by being there).

Replay$_{\rightarrow \mathbf{3.7}}$ can be used by an Interest Agent to provide a very detailed report of the activities.

3.9 Away Message

Context: Users interact in a request-response scheme with differing levels of synchronicity.

Problem: Users expect their interaction partners to respond quickly to their actions, but sometimes an interaction partner is unable to respond quickly. The longer initiating users have to wait, the greater their frustration.

Scenario: Martin has encountered problems using the graphics components of the game engine, so he contacts Carla to ask her a question. Normally, Carla responds after

several minutes, but by the evening of Martin's working day there is still no response.

Symptoms: *Consider to apply the pattern when ...*

- Senders ask recipients whether they received a message because the senders did not get a response.
- Senders wait for a recipient's action, but this action does not happen.
- Senders are used to quick replies from their interaction partners based on previous experience. This means that they expect a specific responsiveness from their interaction partner.
- Users are away from the interaction space from time to time.

Solution: **Let the groupware system respond to a communication with an automatic away message whenever a normal response time cannot be guaranteed. Provide information on when the requesting user can expect a response.**

Collaborations: The AWAY MESSAGE pattern suggests following a three-step process when users leave an interaction context temporarily:

Setup. Before users leaves the interaction space, they think about the duration of their absence. They create an away message that explains why they cannot respond and which includes information on the earliest possible reply (their estimated return date).

In most cases, users also provide an explanation that helps senders to handle urgent requests. An example of this is an explanation about who a sender can contact during the recipient's absence.

Design the set-up process so that it is quick and easy. Ideally a user should be able to activate an away message with only one click. It may otherwise be too time-consuming to set up an away message when a user is about to leave.

Execution. When a sender sends a message to an absent recipient, the recipient's system automatically replies with the away message. To avoid duplicate notifications, the recipient's system in addition remembers that the sender was notified. Further e-mails from the sender will not be automatically replied to.

Tear Down. When absent users return and are ready to reply to messages normally, they deactivate their away messages. Optionally, senders who received an away message can be informed that an addressee has returned.

Danger Spots: One of the largest problems with away messages is that they often do not distinguish between bilateral and group communication. When communicating via a FORUM, for example, which delivers messages to the users' e-mail boxes, the recipient's system may reply with an away message that is received by the whole group instead of the individual sender. The reason for this is that many e-mail dispatching mechanisms modify the message headers so that the sender field is different to the reply-to field. The recipient's system keeps track of message senders, but replies to the reply-to address. This address is again multiplexed to all members of the forum. Forum members may therefore receive multiple away messages, which is annoying.

When discussing this pattern with our copy-editor, he reported a recent case of an employee who had left a company with an active and permanent away message. For many months afterwards every single posting to a mailing list resulted in a reply to each of the several thousand members of the mailing list that stated that "John Doe is no longer at (this company)". Almost no-one on the group had ever even heard of the person or the company involved.

A solution is to keep track of the addresses to which an away message has been sent, instead of the message senders to which the message was a reaction. An alternative solution that is often used is not to auto-reply with an away message if the original message was not personally directed to the absentee. For e-mails, for example, this is the case if the recipient is set to the address of the forum and the member of the forum is only added as a Bcc: (blind) recipient of the message.

A comparable problem is that an absentee cannot know in advance who will receive an away message. Since messages are by default sent to anyone who tries to contact the absentee, the information about their absence can be seen by anybody. This can be valuable information, especially for spam mailers, since it proves that the e-mail address is working and hence valid. A possible solution to this is to define a list of e-mail addresses in advance that may receive an away message.

This leads to a more sophisticated version of the AWAY MESSAGE pattern in which the absentee can define different messages for different groups of senders. If you consider for example the case of a lecturer who leaves university for a month, it makes sense to provide several away messages: one for the students, in which questions on course management might be answered, one for faculty members that discusses issues about project work, and a third for external partners. The problem with such sophisticated messages is it requires more effort to set up the AWAY MESSAGE.

You could think about detecting the absence of a user automatically and providing away messages in this case. However, this may lead to two problems: first, such users will no longer be in control of whether they tell other users about their absence. This violates the users' privacy. Second, such a system will in most cases not know when the absentee returns. This information is, however, needs to be included in effective away messages.

Known Uses:

Vacation [4] is probably the most widely used implementation of an away message. In the activation phase, users can specify a message body that is from then on sent to all senders of e-mail to the absent user. Whenever replying with an away message, the system keeps track of the sender and ensures that no duplicate messages are created.

Instant messaging systems like Trillian `http://www.trillian.cc/` allow users to add a message that explains that they are currently away. When another user tries to initiate a chat communication, the system automatically replies with the away message provided.

Related Patterns:

AVAILABILITY STATUS also helps senders to stay aware of the status of a request. However, the availability status normally does not reveal any temporal estimation of response.

ALIVENESS INDICATOR$_{\rightarrow 3.4}$ An ALIVENESS INDICATOR is an alternative to an away message, since it also signals the requesting users that they may not expect an immediate response. However, it does not explain why the absentee cannot respond, or when the absentee will again be available.

4. CONCLUSION

Within this paper, we presented a set of patterns that address one of the most important aspects of asynchronous group collaboration: staying aware of what has happened in a collaboration space during a longer time of absence. Now that the final version of this paper is made available, we can also look back on 18 months in which these patterns began to influence the construction of collaborative systems.

One example is the XecliP system, an eclipse extension that supports distributed pair programming (`http://sourceforge.net/projects/xeclip/`): Although intended for synchronous interaction, the system allows to REPLAY$_{\rightarrow 3.7}$ interaction that took place in a pair programming session and inspect the interaction between participants, e.g., for reaching a better understanding of the code's evolution. With systems like XecliP, the scenarios described in this paper have become reality, although most distributed teams still ignore most of the potentials for close interaction.

When using the patterns in our teaching, we observed that students become much more innovative in designing collaboration support. In this sense, the patterns of this paper as well as the rest of the pattern language for computer-mediated interaction starts to show its real impact, namely that it helps designers to take new opportunities given by networked collaboration into account and shape the environments of the future in which we will spend an increasing part of our time.

5. ACKNOWLEDGEMENTS

Munawar Hafiz did a great job as a shepherd for PLoP 2006 where these patterns were first discussed. His questions helped us to make the following patterns more clear. The workshop participants shared their experiences with us which helped us to get another step closer to understanding the real forces behind asynchronous interaction. Finally, we would like to thank John Wiley & Sons for allowing us to include abbreviated versions of patterns from [25] in this paper.

6. REFERENCES

[1] R. Bentley, W. Appelt, U. Busbach, E. Hinrichs, D. Kerr, K. Sikkel, J. Trevor, and G. Woetzel. Basic support for cooperative work on the world-wide web. *International Journal of Human-Computer Studies: Special issue on Innovative Applications of the World-Wide Web*, 1997.

[2] J. M. Carroll, D. C. Neale, P. L. Isenhour, M. B. Rosson, and D. S. McCrickard. Notification and awareness: synchronizing task-oriented collaborative activity. *Int. J. Hum.-Comput. Stud.*, 58(5):605–632, 2003.

[3] G. Chung, P. Dewan, and S. Rajaram. Generic and composable latecomer accommodation service for centralized shared systems. In S. Chatty and P. Dewan, editors, *IFIP Working Conference on Engineering for HCI*, pages 129–145, Heraklion, Crete, Greece, 1998. Kluwer Academic Publisher.

[4] B. Costales. *sendmail*. O'Reilly, 3 edition, 2002.

[5] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In *Conference proceedings on Computer-supported cooperative work*, pages 107–114, 1992.

[6] T. Erickson and M. R. Laff. The design of the 'babble' timeline: a social proxy for visualizing group activity over time. pages 329–330, 2001.

[7] A. Fernandez, T. Holmer, J. Rubart, and T. Schümmer. Three groupware patterns from the activity awareness family. In *Proceedings of the Seventh European Conference on Pattern Languages of Programs (EuroPLoP'02)*, Konstanz, Germany, 2003. UVK.

[8] G. Fitzpatrick, T. Mansfield, S. Kaplan, D. Arnold, T. Phelps, and B. Segall. Augmenting the workaday world with elvin. In *Proceedings of ECSCW'99*, pages 431–451, Copenhagen, September 1999. Kluwer Academic Publishers.

[9] G. Fitzpatrick, T. Mansfield, S. Kaplan, D. Arnold, T. Phelps, and B. Segall. Instrumenting and augmenting the workaday world with a generic notification service called elvin. In *Proceedings of ECSCW 1999*, 1999.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[11] C. H. Ganoe, G. Convertino, and J. M. Carroll. The bridge awareness workspace: tools supporting activity awareness for collaborative project work. In *NordiCHI '04: Proceedings of the third Nordic conference on Human-computer interaction*, pages 453–454, New York, NY, USA, 2004. ACM Press.

[12] C. H. Ganoe, J. P. Somervell, D. C. Neale, P. L. Isenhour, J. M. Carroll, M. B. Rosson, and D. S. McCrickard. Classroom bridge: using collaborative public and desktop timelines to support activity awareness. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 21–30, New York, NY, USA, 2003. ACM Press.

[13] E. Gottesdiener. Team retrospectives – for better iterative assessment. *The rational edge*, April, 2003.

[14] J. Grudin. Cscw introduction. *Communications of the ACM*, 34(12):30–34, 1991.

[15] C. Gutwin and S. Greenberg. Workspace awareness for groupware. In *Proceedings of the CHI '96 conference companion on Human factors in computing systems: common ground*, pages 208–209, Vancouver, BC Canada, 1996.

[16] J. Henkel and A. Diwan. Catchup!: capturing and replaying refactorings to support api evolution. In

ICSE '05: Proceedings of the 27th international conference on Software engineering, pages 274–283, New York, NY, USA, 2005. ACM Press.

[17] N. L. Kerth. *Project Retrospectives: A Handbook for Team Reviews*. Dorset House Publishing Company, Incorporated, 2001.

[18] S. Lukosch. *Transparent and Flexible Data Sharing for Synchronous Groupware*. Schriften zu Kooperations- und Mediensystemen - Band 2. JOSEF EUL VERLAG GmbH, Lohmar - Köln, Aug. 2003.

[19] S. Lukosch. Transparent latecomer support for synchronous groupware. In J. Favela and D. Decouchant, editors, *Groupware: Design, Implementation, and Use, 8th International Workshop, CRIWG 2003*, LNCS 2806, pages 26–41, Grenoble (Autrans), France, Sept. 2003. Springer-Verlag Berlin Heidelberg.

[20] A. Sarma, Z. Noroozi, and A. van der Hoek. Palantír: raising awareness among configuration management workspaces. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 444–454, Washington, DC, USA, 2003. IEEE Computer Society.

[21] T. Schümmer. Lost and found in software space. In *Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS-34), Collaboration Systems and Technology*, Maui, HI, 2001. IEEE-Press.

[22] T. Schümmer. GAMA – a pattern language for computer supported dynamic collaboration. In K. Henney and D. Schütz, editors, *Proceedings of the Eighth European Conference on Pattern Languages of Programs (EuroPLoP'03)*, Konstanz, Germany, 2004. UVK.

[23] T. Schümmer. *A Pattern Approach for End-User Centered Groupware Development*. Schriften zu Kooperations- und Mediensystemen - Band 3. JOSEF EUL VERLAG GmbH, Lohmar - Köln, Aug. 2005.

[24] T. Schümmer and A. Fernandéz. Patterns for virtual places. In *Proceedings of the Tenth European Conference on Pattern Languages of Programs (EuroPLoP'05)*, 2005.

[25] T. Schümmer and S. Lukosch. *Patterns for Computer-Mediated Interaction*. John Wiley & Sons, Ltd., 2007.

[26] J. Tidwell. *Designing Interfaces*. O'Reilly, Sebastopol, CA, USA, 2006.

[27] F. B. Viègas, M. Wattenberg, and K. Dave. Studying cooperation and conflict between authors with history flow visualizations. In *CHI'04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 575–582, New York, NY, USA, 2004. ACM Press.