THE ABSENT PARTICIPANT
More patterns for group awareness

Till Schümmer & Stephan Lukosch
FernUniversität in Hagen
Computer Science Department
Informatikzentrum, Universitätsstr. 1, D-58084 Hagen, Germany
till.schuemmer@fernuni-hagen.de

August 26, 2006

Note to the Writer's Workshop: We are aware of the fact that this paper exceeds the size of a paper that can be discussed in depth in a workshop. Therefore, we'd like to ask for feedback on the problem/solotion pairs of all patterns and the full content of the following four patterns: Away Message$_{\rightarrow 9}$, Life Indicator$_{\rightarrow 4}$, Replay$_{\rightarrow 5}$, and Timeline$_{\rightarrow 6}$.
The patterns listed in the appendix are only provided to help the reader to better understand the context of the pattern language. They need not to be in the writer's workshop's attention.

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. This context will be partially summarized in the paper.

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 (Schümmer and Fernandéz 2005) at which they meet at the same or different times. Understanding and better supporting distributed collaboration has been the goal of the research area of CSCW (Computer Supported Collaborative Work) (Grudin 1991). 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 114 patterns in this field (Schümmer and Lukosch 2007). In this paper we will present 9 additional patterns that focus on asynchronous interaction.

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 (1992) 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 (Fernandez et al. 2003), (Schümmer 2004), (Gutwin and Greenberg 1996). For asynchronous collaboration, there are fewer examples in the research literature, however, some practices have proven to work.

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 Hong Kong. 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 African pilot users located in Ethiopia and in Malawi.

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 Hong Kong. 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 Hong Kong 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

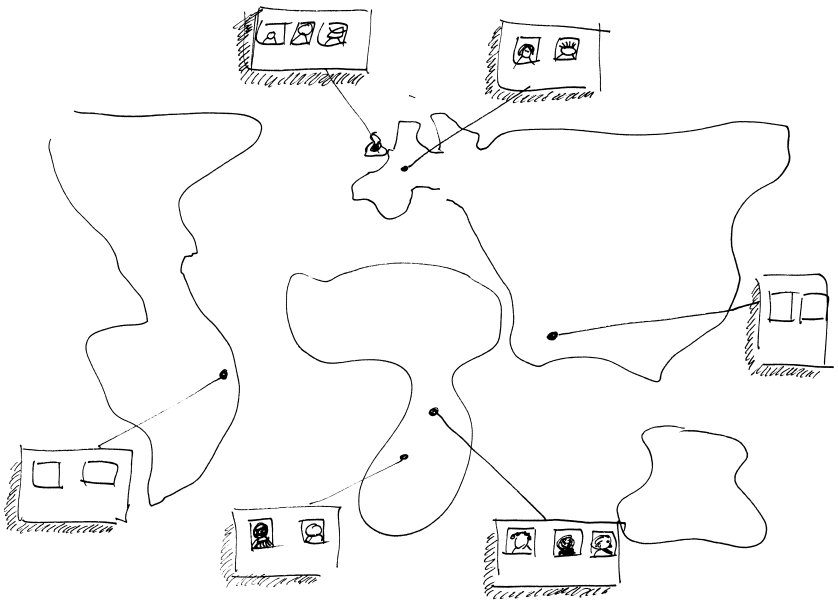


Figure 1: Distribution of team members in the example setting.

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.

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. 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 (Schümmer 2004), we only include their thumbnail in this paper:

Activity Log

Problem: Merging two users' (past or current) work is a difficult task. It requires

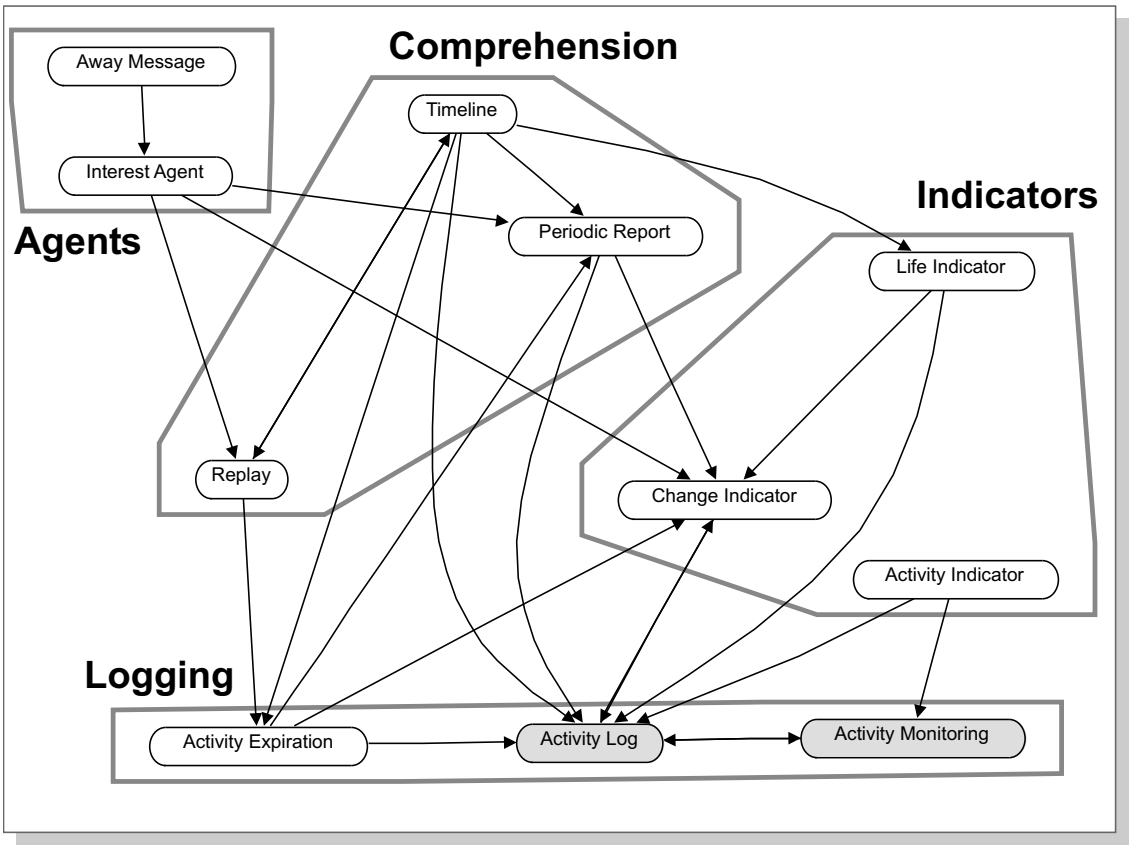


Figure 2: Patterns for asynchronous group awareness described in this paper.

that the activities are transferred to the same context and that the goals are aligned. But many applications don't provide access to the artifact's history, its use, and its evolution. Thus, merging is vulnerable to errors and often collaboration does not take place since the merging efforts exceeds the estimated gains of a collaboration.

Solution: Remember all activities that users perform on shared artifacts – not only modifying accesses, but also read accesses. Provide access to the activities, so that a user can understand (and merge) other users' activities with his own 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$_{\rightarrow}$ or a Local Awareness$_{\rightarrow 10}$) to subscribe to monitored activities.

These two patterns help to capture events. In line with Fitzpatrick et al. (1999b), we define an event as *"any significant change in the state of an observed object"*. The Activity Expiration$_{\rightarrow 1}$ pattern which is added to the *Logging* clus-

ter 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$_{\rightarrow 2}$ shows where the other users have been active while the Change Indicator$_{\rightarrow 3}$ visualizes modifications to artifacts that the absent user has not yet seen. The Life Indicator$_{\rightarrow 4}$ finally shows whether or not a user has been active at all.

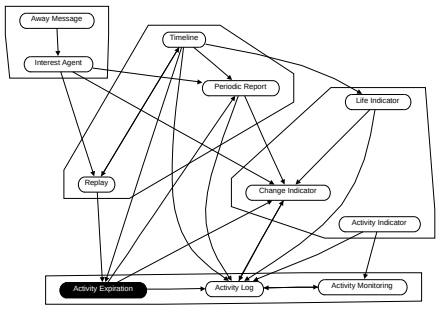
These patterns focus on individual activities. After a longer period of absence, this is not sufficient to get a holistic idea of the group's progress. The three patterns in the *Comprehension* cluster therefore support the user in better understanding sequences of activities. The Replay$_{\rightarrow 5}$ pattern replays activities like in a movie. The Timeline$_{\rightarrow 6}$ provides a quick overview by chronologically visualizing changes. The Periodic Report$_{\rightarrow 7}$ 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$_{\rightarrow 8}$ collects activities that are relevant for the absent user and the Away Message$_{\rightarrow 9}$ 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. Whenever patterns reference other patterns of this language, you can find short versions of these patterns in Section 10.

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.

1 Activity Expiration



Alternative name(s): Remember to Forget

Intent Do not consider activities from an activity log that are no longer relevant to the users.

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$_{\rightarrow 10}$. 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 *You should 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 **Therefore: 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$_{\rightarrow}$ and used to provide awareness on past actions. The pattern proposes to have two trig-

gers 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$_{\rightarrow 7}$, 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 (Bentley et al. 1997) maintains an Activity Log to visualize modified artifacts using Change Indicators$_{\rightarrow 3}$. 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.

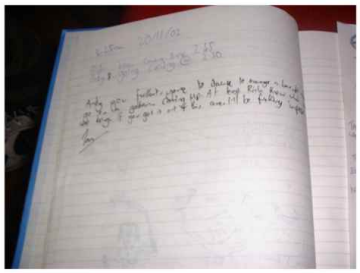
Motorola E1000 is a mobile phone in which the SMS configuration may be appropriated in order to keep only a specific number of short messages and skip old messages.[1]

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.

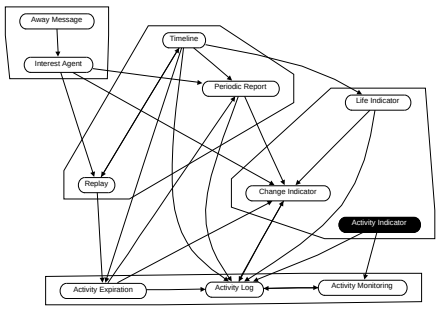
Related Patterns Activity Log$_{\rightarrow}$: The Activity Log stores all activities that were performed in a collaboration space. The importance to consider not all of these activities is raised by the Activity Expiration pattern.

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

2 Activity Indicator



A logbook



Alternative name(s): Ticker Tape

Intent Provide an explanation about other user's activities while not yet showing the activity's intermediate result.

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 the users. In a co-located setting users are used to perceive non-verbal signals such as movements or noises if 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 Paul and Martin both decided to work on graphical aspects of the game engine project. Paul starts to work on the rendering of places while Maurice looks at the rendering of actors. At one point in time, Maurice starts changing some code that he has currently checked out to his workspace. He plans to check the code in again after he has tested his modifications. However, if Paul also wants to modify the same bit of code, the changes of both users will need to be merged after both users have finished their tasks.

Symptoms *You should consider to apply the pattern when ...*

- users dislike distributed interaction as they do not know what the other users are doing.
- users perform concurrent actions.
- users wait too long for another user's action although 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 of staying aware of other users.

Solution **Therefore: Indicate the current activities of the other users in the user interface. To reduce interruptions, use a peripheral place.**

Collaborations Provide an user interface element in a peripheral place that shows that remote users are active and what they are currently doing. Ensure that the remote users' activities are shown in the user interface immediately after they started the action. Hide the activity if no more activities are detected from the remote users (e.g., no keyboard input for a specific period of time).

Rationale The activity indicator shows up immediately after a remote user started an activity. This signals to the local user that the remote user is now active and that additional local actions could lead to conflicts.

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

– Where are you going to display the other users' activities? Commonly used places are
  ▷ the status bar of the collaborative application,
  ▷ the status bar of the desktop,
  ▷ a pop-up note on the desktop's task bar that disappears automatically after a short period of time, or
  ▷ a notification pane in your application (comparable to an Embedded Chat$_{\rightarrow 10}$).
– What kind of activities are you going to indicate? Is it only important to show modifying activities or do you also indicate, e.g., navigation activities of the users.

Danger Spots If many users collaborate, displaying the different activities becomes difficult. Therefore, reduce the provided information and cluster the information, e.g. when several users are performing the same activity.

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 understand if he can expect a reply.

Palantír (Sarma et al. 2003) is a group awareness component that extends a software configuration management system. Palantír tracks the activities of the group members to provide knowledge about the team members' actions.
Each activity that is of interest to the local user runs through a ticker tape of the awareness client. This view is intended to stay on the user's desktop and constantly update the user on the newest activities. Using a scrolling text will probably

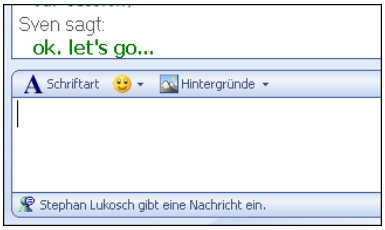


Figure 3: An activity status bar in the MSN Messenger.

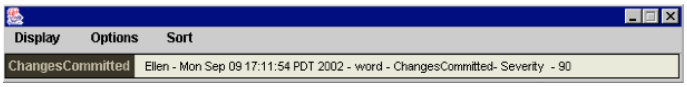


Figure 4: The ticker tape of Palantír.

result i a higher attention level spent on the information.
A comparable visualization is used in the Elvin system (Parsowith et al. 1998) that tracks and visualizes activities in a shared workspace.

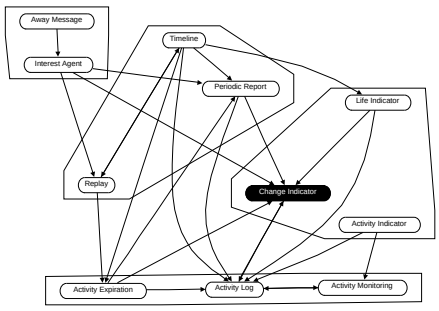
Mail Clients like Thunderbird `http://www.mozilla.com/thunderbird/` provide an status icon whenever a new message arrived. This prevents the local user from polling his mailbox manually. Compared to more prominent forms of new-mail alerts, the icon in the status bar will not distract the user too much: 'By having only the new email icon in the system tray, employees' attention would be attracted only when the concentration level is less demanding and the interruption would occur at a more convenient time.' (Jackson et al. 2003).

Related Patterns

Attention Screen$_{\rightarrow 10}$ can be used to filter the awareness information shown in the status line.

Activity Monitoring$_{\rightarrow}$ describes how users' actions c an be detected at a technical level. Although the pattern has the goal of storing records on the detected activities in an Activity Log$_{\rightarrow}$, one can use the described mechanism as well to inform other users on the detected activity.

3 Change Indicator



Intent Indicate that a shared document has been changed by another user.

Context Users work on independent copies of the shared artifacts.

Problem **While a user works on an independent local copy of the artifact, her checkout frequency may be low. So she may work on an old copy, which leads to potentially conflicting parallel changes. The conflict is worse if two parallel modifications had contradicting intents.**

Scenario Paul has done some major improvements on the graphics engine. The most important was that he changed the coordinate system from a cartesian to a polar coordinate system. Paul documented this change in the manual of the game engine. Martin however was not aware of this change since he knew the game engine before and does not frequently read the manual. Thus, he uses the graphics engine part with cartesian coordinates and is confused that the images look very strange.

Symptoms *You should consider to apply the pattern when ...*

- Users apply changes to artifacts based on their old knowledge of the artifact's state.
- Users tell that they would have done the change differently, if they had known the newer state of the artifact.
- Users frequently change artifacts.

Solution **Therefore: Indicate to the local user whenever an artifact has been changed by another user. Show this information whenever the artifact is shown on the screen. The information should contain details about the kind of the change and access to the new version of the artifact.**

Collaborations Figure 5 shows how the different participants collaborate. User 1 checks out 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 an independent copy of the document A which has 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 of document A. As user 2 has not modified the local copy of document A, she checks out the new version, before performing her planned modifications.

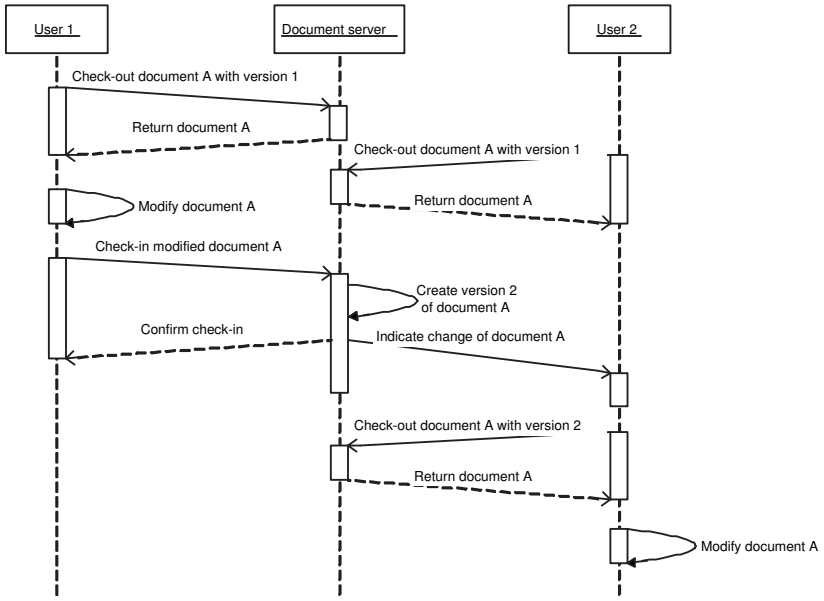


Figure 5: Two users are working on a shared document

Rationale

There are two main reasons, why the pattern works: a technological reason and a cognitive reason.

From a technical point of view, indicating changes alters the point in time, when integration is performed. Whenever an artifact was changed, all older versions will be marked to tell the user that the artifact was changed by another user. In most cases, the second user will integrate the change of the first user immediately to base his changes on the most recent version. If this is not possible, he can at least inspect the newer version and model his own change in a way that an integration is easy. This reduces the cost of integration. He can also get in contact with the person, who applied the first change (remember to respect the user's privacy and smoothly approach him using an Intimacy Gradient$_{\rightarrow 10}$). Both users can then discuss and align their changes and – if considered useful – work together in a tightly coupled mode. In any of the mentioned ways, the cost of integration is reduced because there have not yet been conflicting changes.

The cognitive reasoning is often much more important. Consider a system, where artifacts are not explicitly stored in a local workspace. At a first glance, such systems do not fit into the context of this pattern. But if one takes a closer look on the interaction, one can define an implicit local workspace: the local user's knowledge because he remembers how the artifacts look like. Whenever an artifact is perceived by the local user, it leaves traces in his memory. All future activities on shared artifacts will be influenced by these memory traces. In many cases, the user will be confident to know a specific artifact and thus not look at this artifact again.

When the artifact was changed, it is important to inform the user that he can no longer be confident in knowing the artifact. The version of the artifact that he remembers can lead to other interpretations than most recent version. The user therefore needs to reprocess the changed artifact and update his semantic representation of the artifact and its context.

Check

When applying this pattern, you should answer these questions:

- How do you visualize the change warning? Can you add a decorator to the icon representing the changed artifact?
- How important is the awareness of the changed artifact? In case of low importance, you can consider to simply change the artifacts visualization after another user changed it while in case of a high importance, you should consider a more obtrusive change indicator such as a dialog window.

Danger Spots

- The calculation of conflicting activities may be complicated if users are allowed to select their desired version (which is true for most environments). Consider the case where a user Alice performed a change on an artifact. Later on, she notices that her change was not right. She thus loads the older version of the artifact. But the Activity Log$_{\rightarrow}$ still remembers the activity that described the change for the artifact. This activity will be considered as a conflicting activity even for Alice (since she now works on a version, which has been changed in between). But the version, which was created by the activity is no longer used by any user. Thus, one should ignore these activities when calculating conflicting activities.
- Even when indicating changes to the users, the users might ignore the indications. To overcome this problem, it is necessary to establish a social protocol for the collaborating users which defines how Change Indications must be handled.
- Changes can be complex and providing details about the can be complex, too. In that case, consider to provide a comparison view for the local user's state of the artifact with the remote user's state of the artifact.

– If changes occur too frequently, most artifacts will be visualized as changed artifacts. The user will then constantly look for the changes and might not find time to perform constructive actions. One solution could be to indicate that an artifact has not been changed allowing the use to skip processing this artifact again.

Known Uses

TUKAN: The programming environment TUKAN (Schümmer 2001) uses a weather metaphor to display change warnings. A heavy lightning symbol tells the programmer that a specific artifact has been changed. The symbol shows better weather for possible conflicts caused by changes on artifacts that are further away (following the ACTIVE NEIGHBORS$_{\rightarrow 10}$ pattern). If there is no near conflict, a sun is shown to indicate that everything is up to date and confirm the user's self-confidence.

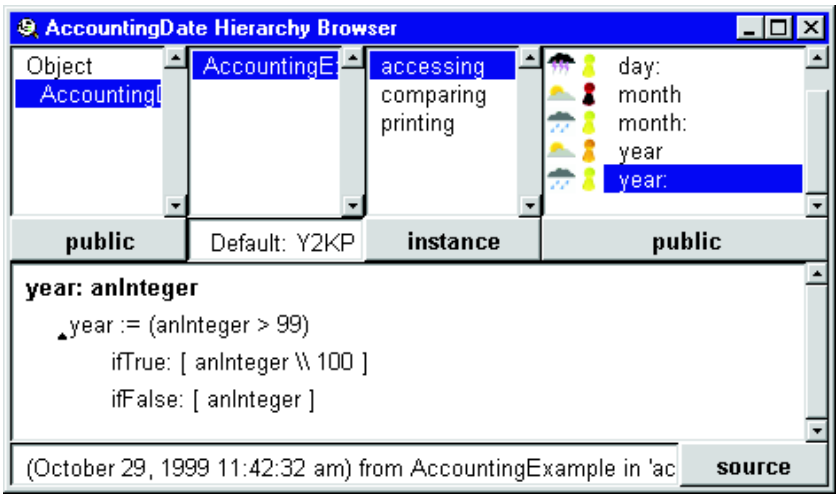


Figure 6: Change-warnings in the collaborative software development environment TUKAN.

Figure 6 shows a browser in TUKAN, where the method `day:` was changed by another user (indicated by a heavy lightning symbol in front of the method name).
By indication of possible configuration conflicts, parallel changes of the same artifact can be avoided. Changes made by other programmers are not instantly reflected in the local programmer's code, but rather in the visualization of the method identifier. Whenever a newer version is signalled, the user may decide to integrate this version before she changes the artifact itself and thus avoid parallel versions.

WinEdt: The text editor WinEdt (`http://www.winedt.com/`) as many other editors buffers the current file in memory, while the user performs edit operations on the possibly shared file. When another user or application changed the file, it displays a

warning that informs the user that the current file was modified (fig. 7).

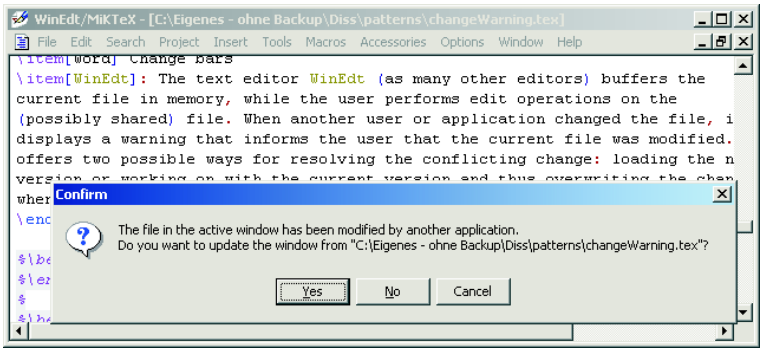


Figure 7: Change-warnings in single user applications.

It offers two possible ways for resolving the conflicting change: loading the new version or working on with the current version and thus overwriting the changes when the document is saved.

BSCW (Bentley et al. 1997) is a shared workspace system that displays change indicators for documents that were modified since the last visit. Although it does not model an explicit local workspace, it remembers, which artifacts have been read by a specific user. These versions of the artifacts form the user's implicit local workspace. The change indicators then lead the user to new or changed information on the BSCW server.

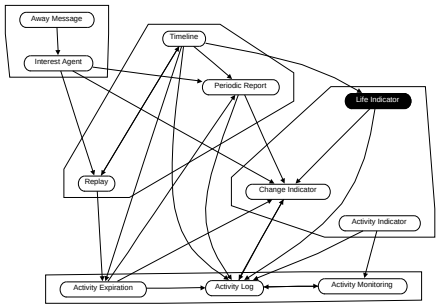
Related Patterns

ACTIVE NEIGHBORS$_{\rightarrow 10}$ should be used, if artifacts are semantically related. In this case, it is important to inform a user not only on changes on the current artifact, but also on changes that might have an impact to the current artifact (on a semantic level).

ACTIVITY LOG$_{\rightarrow}$: Use an ACTIVITY LOG to store the activities that are used for calculating conflicting activities.

PRESENCE INDICATOR$_{\rightarrow 10}$: The presence indicator is comparable to the CHANGE INDICATOR with respect to the fact that it also visualizes activities on artifacts. The main difference is that PRESENCE INDICATORS only consider activities that are still active. In most cases, CHANGE INDICATORS inform on activities that are completed.

4 Life Indicator



Alternative name(s): Virtual Tamagotchi

Intent — Include an indicator in a virtual environment that reflects a user's activity.

Context — User 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. Especially, they cannot easily see whether other users were active during their absence. This makes it hard to experience the group's progress.**

Scenario — Maurice 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 Paul. However, he does not know whether or not Paul has also done some work in this area while Maurice was out of office.

Symptoms — *You should consider to apply the pattern when ...*

- users complain that other users have stopped to participate although these users still follow the interaction.
- users ask the whole group to state whether or not they still participate.
- users ask other users to visit the collaboration space but are not sure whether or not the other users followed their request.

Solution — **Therefore: Show a Life Indicator together with the user's virtual representation. If a user participated in the collaboration space recently, use a picture for the indicator that looks very *alive*. Use gradually less *lively* pictures to represent the period of inactivity. Let the picture look like something for which the user can take responsibility.**

Collaborations

The system keeps track of a user's last activity 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 from now to the time of the user's last activity. Depending on the length of the time span, the system selects a different indicator that is shown together with the user's representation. In cases of a short time span, the indicators connotates a high level of life while a long time span is visualized by a less lively picture.

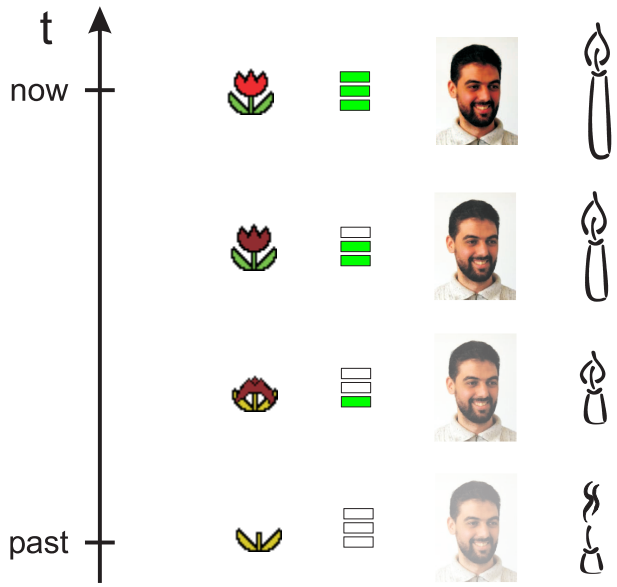


Figure 8: Examples for life indicators.

Figure 8 provides examples for life indicators: a withering flower, a declining bar chart, a fading picture, and a candle that burns down.

Optionally, the indicator provides additional information (e.g., using a tool tip). It can show the time when the user was active or it can provide details on what the user did when he was active. The indicator can 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 takes care of (e.g., the flower in Figure 8). In systems where the collaboration space is less important, there can be one global scope. This means that the user has the same indicator in the whole system. The fading user image is one example for such a visualization.

The second decision for the scope is whether the indicator is bound to an individual or to an artifact or a region in the collaboration space. An example for the latter is that the place has a virtual flower and that every activity in the place waters this flower.

Rationale

The knowledge of other users' latest time of participation helps to better understand how close the group is collaborating. Especially, it shows if there are users who have not been present in the workspace for a long time. Since all group members see this information, the group can think about ways to contact the absent user, analyze why the user did not participate and reassign the group's tasks accordingly.

Having indicators at different scopes can support the group to distinguish between the situation where a user was offline for a while from the situation where a user was engaged in a different space. Again, this can help to better understand the user's context.

Finally, the indicator can motivate group members to at least enter the collaboration space in order to keep their indicators alive (which could be considered as a *tamagotchi effect*[2]).

Danger Spots

- The mentioned *tamagotchi effect* can be problematic since users could only pretend to participate.
- The indicators can create a large social pressure to participate. In contexts where the participation is less mandatory, you should think about using the MASQUERADE$_{\rightarrow 10}$ pattern to allow users to turn their indicator off. In any case, this should be coupled with the RECIPROCITY$_{\rightarrow 10}$ pattern so that users who turn their indicator off cannot see other users' indicators.
- Using two many life indicatory, i.e., at every place in the collaboration space, can lead to a situation where the users constantly chase their indicators so that they stay alive. Consider, e.g., the example of the virtual flower that is bound to a workspace. If a user is allowed to enter many workspaces, he would also have many flowers and he would need to look for them even if the workspace was deserted for a longer time. A solution to this problem could be either to reduce the number of places that have flowers for the user or to slow down the ageing of the flowers so that they only fade when other users are active and the owner of the flower is absent. This however complicates the calculation of the flower's age.

Check

When applying this pattern, you should answer these questions:

- What metaphor do you use to visualize the life indicator?
- Will you support different scopes? What are the scopes?
- Should the life indicator be bound to people or to collaboration spaces?
- What is the time scale for your indicator? This is dependent

[2] A Tamagotchi (see `http://www.tamagotchieurope.com/`) is a popular digital toy that was invented in 1997 by Aki Maita. The idea of the toy is that a small creature is living in an agg-shaped computer that the player can carry along with him over several weeks. The player can feed the Tamagotchi or play with it. Depending on how *well* the player took care of his tamagotchi over several weeks, it evolves to other creatures or died.

on the level of synchronicity that you want to communicate. In asynchronous systems, this may be a week while in more synchronous systems several minutes of inactivity can result in a faded life indicator.

- Will you provide context information?
- Where will you include the indicator? Which parts of your collaboration space should be free of indicators?

Known Uses

Flowers in CURE: In the web-based collaboration space (Schümmer 2005), flowers are used to visualize when a user was active in a group's collaboration space (fig. 9).

Figure 9: Flowers and fading user pictures in CURE.

Each space has a property page that lists all users who are allowed to enter the space. The property page shows the user's picture together with a flower, the user's name, and the user's rights in the collaboration space. Whenever a user performs an activity in the space (e.g., reads a page or modifies content), her flower is refreshed so that it blooms. The flower gets older over time until it has no head anymore (after 7 days).
In Figure 9, the users *Till Schümmer* and *Thomas Jarmer* have been active recently. *Stephan Lukosch* has a slightly darker flower showing that his activity is a short while ago. *Stephan Frysch*'s flower went limp and the flowers of *Petra Kösters* and *Mike Wilhelm* lost their head visualizing that these users were inactive for a long time. The flowers provide context information stating the last time when the user was active.
CURE also shows the global activity of the user by means of fading user pictures. For example, in Figure 9, the user *Stephan Frysch* has been inactive in the shown place but his user picture is still colorful. This implies that he is currently working in another space.

openBC `http://www.openBC.de` is a social networking system that shows an *Activity Meter* on each user's contact page. This represents how active a user has been recently.

Figure 10: An activity meter at openBC.

Related Patterns

ACTIVITY COUNTER$_{\rightarrow 10}$ collects the overall number of activities that took place at a specific location. In contrast to the LIFE INDICATOR it does not show when these activities took place.

CHANGE INDICATOR$_{\rightarrow 3}$ depicts a modification on the selected artifact. The difference is that only modifying activities are shown and that time in most cases does not play an important role in the CHANGE INDICATOR pattern.

ACTIVITY LOG$_{\rightarrow}$ allows to collect activity information about users and thus to calculate how the LIFE INDICATOR has to be visualized.

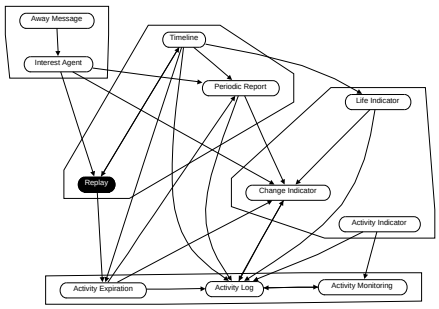
HALL OF FAME$_{\rightarrow 10}$ collects the most prominent places where activities took place.

PRESENCE INDICATOR$_{\rightarrow 10}$ helps to better understand the presence of a user. The LIFE INDICATOR is also applicable in semi-synchronous or asynchronous settings.

USER GALLERY$_{\rightarrow 10}$ provides a place to integrate a LIFE INDICATOR.

VIRTUAL ME$_{\rightarrow 10}$ can be combined with LIFE INDICATOR. Whenever other users browse the page describing the user, they can also see how alive this user is.

5 Replay



Alternative name(s): What Has Happened Here?

Intent Replay how the current state of a collaboration has evolved to update latecomers.

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

Problem **When users join a collaboration as a latecomer 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 session.**

Scenario Paul and Susan performed an extremely successful pair programming session. On the next day, Susan wants to continue the session but Paul caught the flu and cannot participate in the session. Therefore, she asks Liam. Unfortunately, Liam does not know what has been going on in Paul's and Susan's session which gives him a hard time to start collaborating with Susan.

Symptoms *You should consider to apply the pattern when ...*

- users often join a collaboration as latecomer, i.e. they have problems to participate in a collaboration from its beginning.
- users have problems to understand how the current state of a collaboration has been reached.

Solution **Therefore: Capture all changes to the shared objects that are used in the collaboration. When a user joins or rejoins a collaboration, replay the captured changes to show the user how the current state of the collaboration has been reached.**

Collaborations To replay how the current state of a Collaborative Session$_{\rightarrow 10}$

has been reached, it is necessary to keep a log of all changes that were applied to the shared state.

Two cases have to be distinguished, with or without existence of a central server. In the simplest case, there is a well-known site that can be used as a provider for the latecomer. This is, e.g., the case when using Centralized Objects$_{\rightarrow 10}$ but also when using Mediated Updates$_{\rightarrow 10}$ to communicate state changes. In both cases, the well-known site, i.e. the central server or the mediator, has to be enhanced to keep a log of all state changes that were already applied to shared state. Depending on how state changes are distributed, this might either be a log of Distributed Commands$_{\rightarrow 10}$ or a set of Immutable Versions$_{\rightarrow 10}$ for all shared objects.

When a central server is available, the latecomer can contact this server as provider and request the log of state changes. When the latecomer has contacted the provider, the provider includes the latecomer in the complete communication concerning state changes. From that point of time, the latecomer buffers all messages concerning state changes.

After including the latecomer in the communication, the provider supplies the latecomer with log of state changes. As soon as the latecomer has received the log, it starts to re-execute all state changes and visualize the execution in the user interface. While re-executing the state changes, the latecomer displays a control panel that allows the user to set the speed that is used to re-execute the state changes. After replaying the log, the latecomer checks if it received state changes. If there state changes in the buffer, the latecomer also re-executes these state changes. 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, there is not necessarily a participant that has participated in the session from its beginning and thereby knows all state changes. To solve this issue, all or a subset of all clients have to keep the log of state changes, e.g., as a set of replicated objects (Offline Replication$_{\rightarrow 10}$). Then State Transfer$_{\rightarrow 10}$ can be used to provide the latecomer with a copy of the log.

Rationale

By keeping a log of all state changes, transmitting this log to latecomer, and re-executing the state changes in a speed selected by the latecomer, the latecomer can perceive how the current state of the session has been reached.

Check

When applying this pattern, you should answer these questions:

- How does the latecomer identify the provider? Is there a site that is well-known to all latecomers?
- Will users be able to control the replay? Will they see all

changes? Will they be able to adjust the replay speed?

Danger Spots

- Transmitting the complete log of state changes might cost a lot of time, especially when the latecomer is joining very late in the session. To overcome this issue, make regular copies of the shared state and let the latecomer choose the point of time at which the replay has to start. Then select the copy of the shared state that is closest before the point of time selected by the latecomer. Transmit this copy and only the commands that have been executed afterwards to the latecomer. The latecomer uses the copy to initialize the shared objects and starts re-executing the commands from the selected point of time.
- If the collaboration is going on for a long time, the size of the log might require a lot memory. To overcome this issue, reduce the size of the log by compressing it. Another possibility is to make a snapshot of the shared state and remove all changes that led to this state from the log. Then latecomers can only request a replay from the last snapshot.

Known Uses

Collaboration Bus (Chung et al. 1998) is a groupware development environment that offers a service that allows to replay how the current state of a session has been reached. The service is based on a latecomer accommodation server which is called the *logger*. At the site of a participant, a so-called *loggable* captures all events that change the local user interface and sends these events to the logger. So the logger is informed about all changes that a client applies to the user interface of a shared application. When a latecomer wants to join a session, the logger replays all logged events to the latecomer's loggable. Based on these events, the latecomer's loggable creates the user interface. As the log can become very large, the system uses log compression techniques. These compression techniques depend on semantic information about the events that a loggable has to provide. Instead of replaying all events, e.g. mouse movements, the logger can provide the latecomer with the events that resulted in a state change.

DreamObjects (Lukosch 2003a) (Lukosch 2003b) is a groupware framework that keeps the log as a replicated object. Thereby, it does not need a central server as provider of the log. When joining users can choose how many percent of the current should be replayed and the delay between visualizing two different state changes. Figure 11 shows a sequel of screenshots taking while a latecomer was joining with a replay.

ReplayKit (Manohar and Prakash 1995a) (Manohar and Prakash 1995b) is a groupware environment that encapsulates and records a single-user in a so-called session object. Users collaborate asynchronously by annotating, by modifying, and by

exchanging these session objects. The runtime system allows to replay a session object at different speeds according to a user's requirements.

CatchUp (Henkel and Diwan 2005) is a plugin for the Eclipse development environment that allows to record and replay refactorings.

Related Patterns

Centralized Objects$_{\rightarrow 10}$ allows to choose the central server as provider for the latecomer.

Collaborative Session$_{\rightarrow 10}$ allows users to plan and coordinate synchronous collaboration. Users that join a session not on time are called latecomer and need the current state of the session.

Distributed Command$_{\rightarrow 10}$ allows to encapsulate state changes.

Immutable Versions$_{\rightarrow 10}$ allows to identify different versions of a shared object and thereby implement a set of versions.

Mediated Updates$_{\rightarrow 10}$ allows to choose the mediator as provider for the latecomer.

Speed Replication$_{\rightarrow 10}$ or Offline Replication$_{\rightarrow 10}$ can be used to keep a replicated log of state changes.

Activity Expiration$_{\rightarrow 1}$ describes how to reduce the size of the log.

Peer-to-Peer Update$_{\rightarrow 10}$ requires that all or a subset of all clients keep the log of state changes as the participants of a Collaborative Session$_{\rightarrow 10}$ communicate in a P2P network and thus there is no central server that can be chosen as provider by the latecomer.

State Transfer$_{\rightarrow 10}$ directly transfer the current state to a latecomer.

Timeline$_{\rightarrow 6}$ shows the orchestration of different activities in a diagram. The Timeline can be compared to a script of activities, while Replay executes or animates the script.

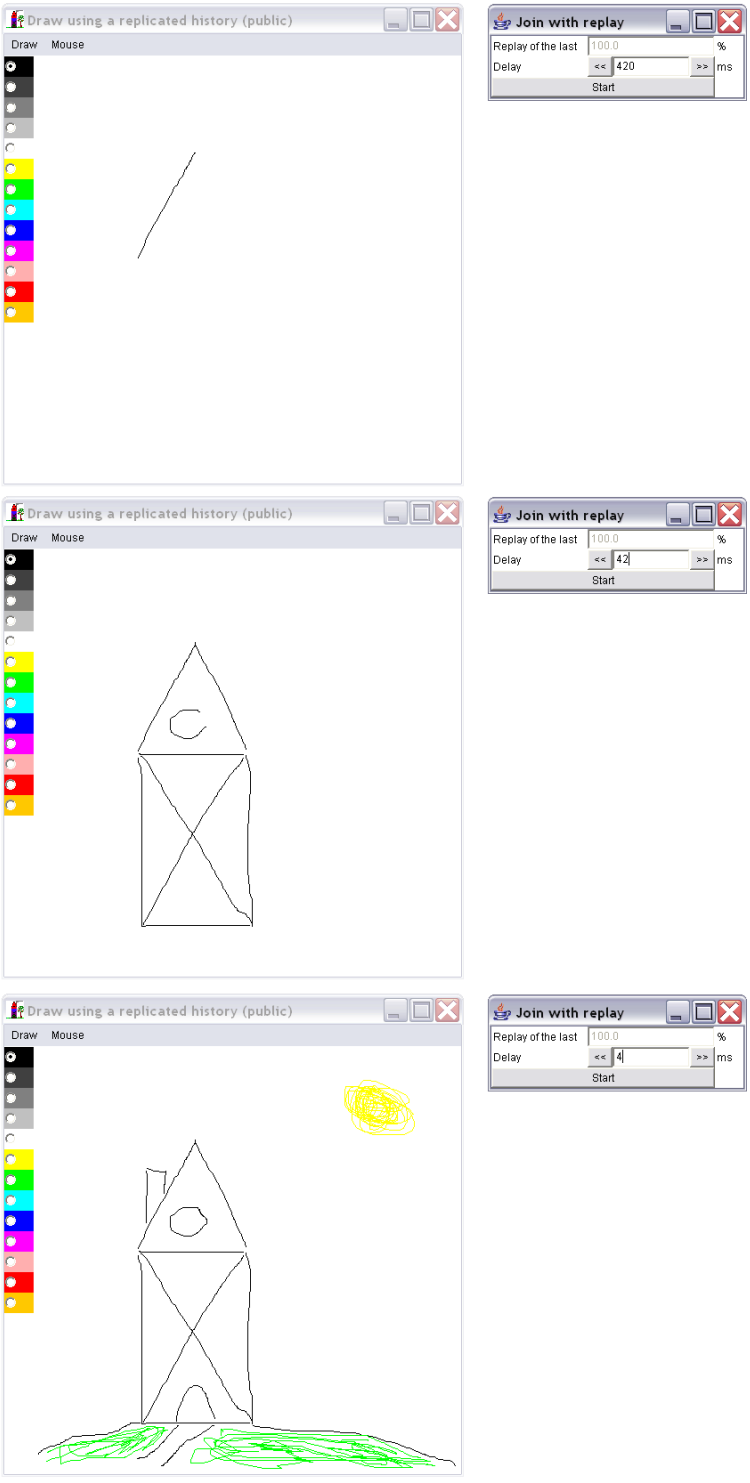
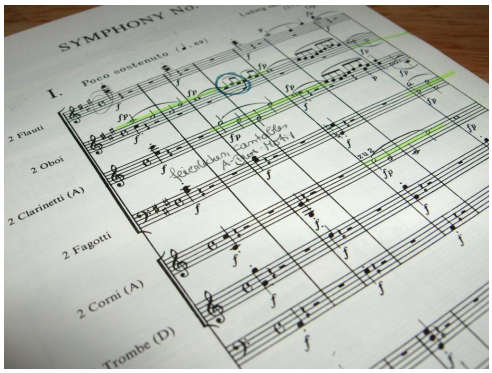
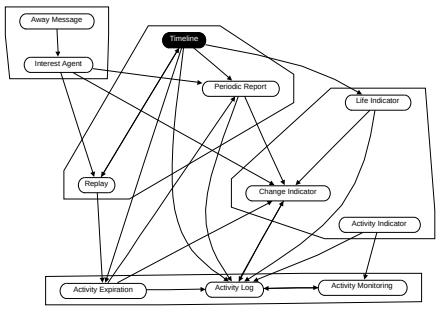


Figure 11: Replay in DreamObjects

6 Timeline





Intent Visualize who has been active at a specific point in time.

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

Problem **Participation in a groups is different. This makes it hard to understand who is working with whom on what topic. But without such an understanding, users lack 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 among them. He decides to examine the change notifications mailed around but even after this he still lacks a holistic picture of the development activities.

Symptoms *You should consider to apply the pattern when ...*

- users complain that other users do not participate although they participate.
- users stop participating but this is not detected by the group.

Solution **Therefore: Visualize the activities of a workspace in 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$_{\rightarrow}$ by the days on which the activity took place. Then show for each day the activities that took place on that day. Separate the different days using bars.

Visualize 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 in one of the axes. When showing the users

in 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 (Tidwell 2006) to support the visualization of long activity logs with many artifacts. This means that additional information is provided on request. Connect the visualization of the activities with the documents that were accessed so that the timeline can be used for navigating to shared objects.

Rationale

The visualization of the group's activities supports the users in understanding the group's actions. It helps him to understand on what topics the users were acting and to what extent they were collaborating. It also helps the user to see who has been active in which area of the collaboration space.

Interaction between users and subgroups can often be recognized when artifacts are shown in one of axes and these are accessed by different participants. The same is true vice versa when the users are shown in one of the axes.

Check

When applying this pattern, you should answer these questions:

- Are you more interested in people or in artifacts, i.e. are you going to show the users or the artifacts in one of the axes?
- How do you code the elements in the diagram? How many different targets will your activities have? Is ist suitable to use color codes or different icons?
- Can you provide tool tips that provide details on the activities shown in the diagram?
- Can you link the points in the diagram to the artifacts or the performers of the activity?
- Does it male sense to distinguish between modifying and reading activities (e.g., by using a different color of the diagram entry)?
- Can you support zooming of the timeline or will it be a static image?

Danger Spots

Ensure that you select the data set with the higher cardinality for the y-axis. If your group, e.g., has five members who work on 50 documents, the documents should be shown on the y-axis and the members should be visualized using different colors or icons. This leads to a diagram with 50 lines and five different icons. Otherwise, one would have a diagram with just five lines but 50 different icons that will probably be harder to distinguish.

One problem of the pattern can be the scalability. As Ganoe et al. (2003) evaluated in a field study, the time line can become "less effective (and even cluttered) if there are frequent changes to all the documents."

Known Uses

Babble Timeline (Erickson and Laff 2001) shown in Figure 12 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.

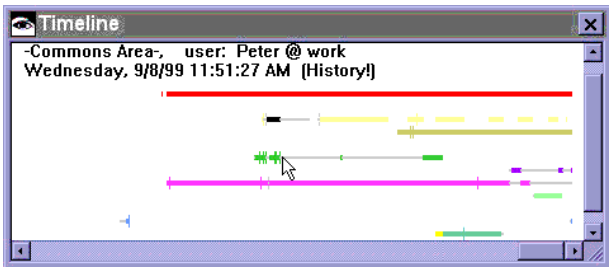


Figure 12: TIMELINE in the Babble chat environment (from Erickson and Laff (2001)).

Each user is represented in a row. When users send messages to the chat, thy leave a peak on their row of the time line. Different colors are used to distinguish contributions to the currently viewed chat from contributions to other chats. Additional information on the time of the contribution and the identity of the user is provided by means of tool tips.
The Babble time line was evaluated with mixed results. Some users reported that they had problems understanding the time line and that they would not use it on a regular basis. On the other hand, a small set of users reported that the time line gave them valuable insights on the "heart beat" of the community: *"I noticed that I missed <userl> by an hour on Monday morning .... <User 2> comes in every so often as a blip. <User 3> jumps from space to space...."* (Erickson and Laff 2001). This user comment reflects a user with exactly the problem addressed by the TIMELINE pattern.

Virtual School is a collaboration space for student interaction. In a user study (Carroll et al. 2003) the authors of the Virtual School environment found several collaboration breakdowns that had their reasons in a lack of activity awareness. One Solution was to integrate a time line as shown in Figure 13 to the students' workspace (the resulting system was then called the BRIDGE awareness center (Ganoe et al. 2004)).
For each project, the time line showed the different documents in the rows of the timeline. Changes to the documents were represented by the icons on the time axis. To access documents, the users could no longer select them from a list of documents but had to select them in the time line instead. Each version could be accessed by clicking on the different icons for a document (using IMMUTABLE VERSIONS$_{\rightarrow 10}$). This made the time line an integral part of daily work.

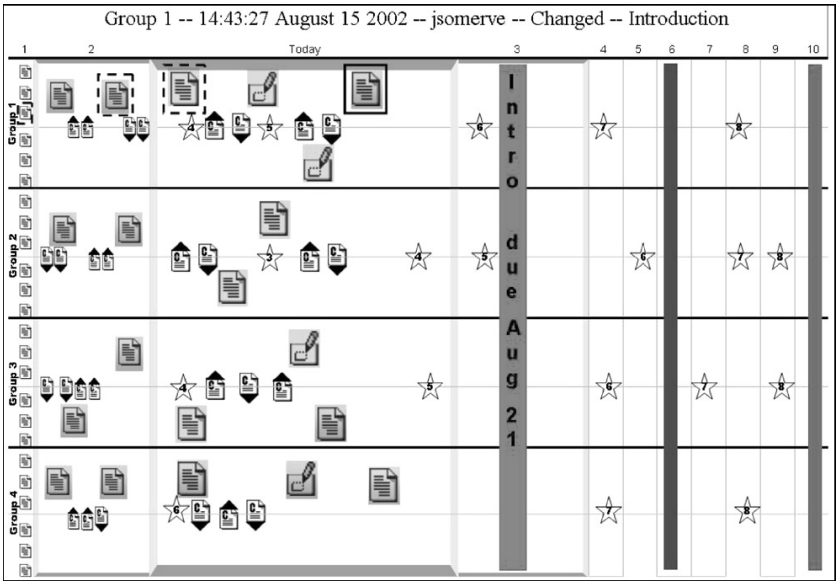


Figure 13: Timeline in the Virtual School context (from Carroll et al. (2003)).

In addition to achieving history awareness, the timeline could be used to plan the future mile stones of the project.
User studies (Ganoe et al. 2003) have shown that the timeline was of high value for people observing the group progress. When there were, e.g., white areas on the time line, teachers queried the students responsible for these documents about problems in their group process and provided help.

Related Patterns Replay$_{\rightarrow 5}$ also addresses the problem of explaining an absent user what activities took place in the collaboration space. The difference is that the Timeline visualizes the activity information, while the Replay pattern shows what the activities changed.

Periodic Report$_{\rightarrow 7}$**:** The Periodic Report provides a more detailed view on changes in a collaboration space. It is well suitable for short time spans but will become very complex when it shows a longer period. The Timeline further abstracts from the activities and is therefore capable of providing a larger overview of activities.

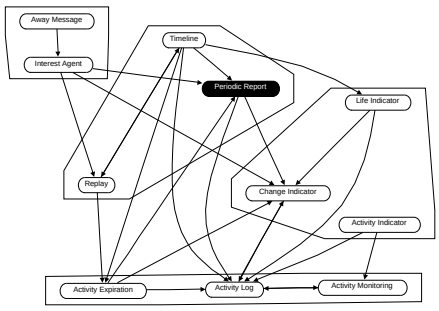
Activity Log$_{\rightarrow}$**:** The Timeline visualizes the activities stored in the Activity Log.

Life Indicator$_{\rightarrow 4}$ helps to detect that a user stopped to participate. This can be seen in the Lifeline when there are no more entries for a specific user (especially when the visualization shows the users on the y-axis).

IMMUTABLE VERSIONS$_{\rightarrow 10}$**:** 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 (which is described in the IMMUTABLE VERSIONS pattern).

ACTIVITY EXPIRATION$_{\neg 1}$ is used to discard activities that are no longer required This can ensure that the timeline does bot get too long.

7 Periodic Report



Alternative name(s): Change Report, Newsletter

Intent Inform the user about changes on relevant artifacts in a user defined frequency (e.g., once a day).

Context Users collaborate asynchronously by modifying shared objects.

Problem **Changes in indirect collaboration are only visible by inspecting the changed artifact. Users want to react to other users' actions but they cannot predict when these actions take place.**

Scenario In January, Weigang has filed a bug report on a security problem of the login mechanism of the game engine. From then on, he frequently checked the affected components for an update. In January he did this daily since he really needed the security fix. But since nothing happened, Weigang reduced the frequency of update checks. Now, 4 months later, he only scans the files every fortnight and does not dare to hope for a fix. Reflecting on the last months, Weigang regrets that he spent so much efforts on looking for changes.

Symptoms *You should consider to apply the pattern when ...*

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

Solution **Therefore: Inform the user periodically about changes that took place between the last notification and the time of the current report.**

Collaborations

The user defines an interest profile manually or automatically based on his access rights in the collaborative system. He also defines a notification interval and a communication channel by which he would like to be notified.

After the interest interval has passed (in most cases at night), the system checks if artifacts matching the interest profile 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 user using the provided communication channel.

Meta information can, e.g., contain a short description of the artifact, information regarding the person who modified the artifact, and the time and type of the modification. It should include a quick reference to the changed artifact to ease the process of accessing it.

The check for changed artifacts can take place in two alternative ways: (1) the system can query the time stamps of all objects and search for those time stamps that fall in the notification interval. This has the advantage that the artifacts only have to carry a time stamp and no additional data structures are needed to track changes. Information regarding the performer of the change needs to be stored with the artifact if such information is desired in the periodic report. Alternative (2) is to track all changes in an Activity Log$_{\rightarrow}$ and query the Activity Log for activities that took place in the notification interval. Since the activities carry all required meta information (performer, time stamp, and type of the 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.

Rationale

Users get informed on changes. This allows them to react to other users' actions. The fixed interval of change notifications ensures that other users can predict when the local user will read the changes. Compared to immediate change notifications the interval reduces the number of interruptions. Since users are able to tailor their report it will only contain relevant information. For that reason, the user will probably read the report.

Check

When applying this pattern, you should answer these questions:

- How frequently do changes occur and how long do users accept to wait until first reactions?
- How can you represent artifacts in the report medium (e.g. in a mail)? Can you provide URLs that let the user access the specific artifact by just one click?
- What is the best time to send your report? Is there a period of low system use in which the report can be sent?
- Should users have to register for the report or do you send it

automatically (consider SPAM concerns)?

- How can users turn the report off?
- How do you support your users in specifying their interests?
- Is the report personalized (because of different interest profiles) or will all users share the same report?

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, users could think that the report got lost if they don't receive any report.

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

Known Uses

BSCW (Bentley et al. 1997) sends reports by e-mail after each day on that the content of a workspace to which the user has access changed or users were active in the workspace (reading).

Figure 14: A daily report in the BSCW Shared Workspace System.

The user can define which types of events (e.g., move events, read events, or create events) should appear in his daily report and which events should be sent immediately (by individual e-mails).

e-Commerce Web Sites often allow users to store their interests and their e-mail address on the server. Whenever a new item is added to the system or an old item is changed, the system sends a notification mail informing the potential customer that there might be a an opportunity for a customer-vendor interaction (e.g., changes in relevant auctions at eBay shown in Figure 15 – `http://www.eBay.com`).

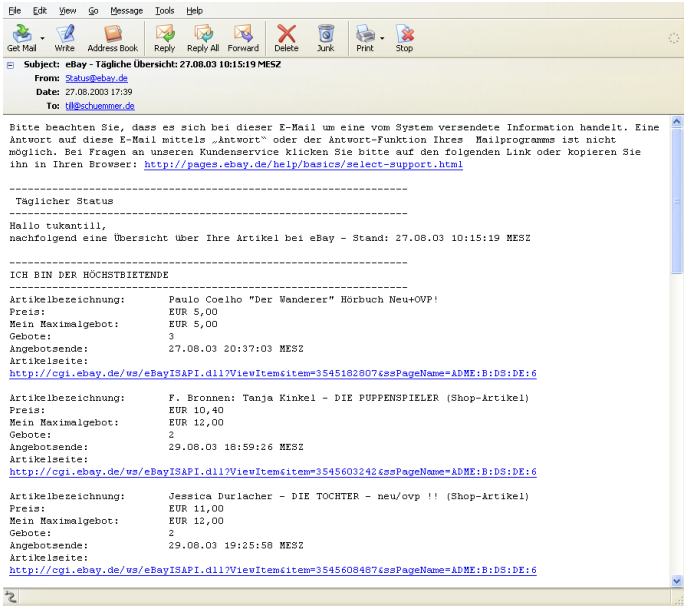


Figure 15: A daily summary sent by the auction platform eBay – **ToDo: Substitute with English Screenshot**.

Mailing lists like the Yahoo Groups (`http://groups.yahoo.com`) often provide options for controlling the frequency of messages sent to the user. The user can decide if he wants to receive individual messages or periodic reports. The reports can contain all message contents or just the headers with links to the individual messages in the web-based mail folder.

Related Patterns

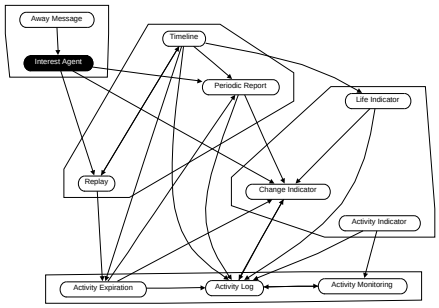
ATTENTION SCREEN$_{\rightarrow 10}$: An attention screen filters notifications and contact requests in order to ensure a user's privacy. It can be combined with the PERIODIC REPORT to ensure that the user stays informed on the activities in the collaborative environment. It can also enhance the acceptance of the PERIODIC REPORT since it allows the users to define which information should reach them via the periodic report.

CHANGE INDICATOR$_{\rightarrow 3}$: The change indicator provides information on changed artifacts in the same context as the artifact itself. The notification that the artifact has changed is attached to the artifact. In contrast, the PERIODIC REPORT ex-

ternalizes this information and transmits it to the user's work context outside the system (e.g., the user's mail box).

ACTIVITY LOG$_{\rightarrow}$: The ACTIVITY LOG keeps track of all activities in the system. The PERIODIC REPORT can be generated from the ACTIVITY LOG by querying it for activities that took place on relevant artifacts in the period of the last report interval.

8 Interest Agent



Intent Collect relevant group activities for an absent user.

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 *You should 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 **Therefore: 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 (VISIBLE AUDIENCE$_{\rightarrow 10}$).

From a technical perspective, the interest agent is an OBSERVER (Gamma et al. 1995) 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$_{\rightarrow 7}$) 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$_{\rightarrow 7}$ up to communication content or navigation activities that are reported by using REPLAY$_{\rightarrow 5}$. 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 (Fitzpatrick et al. 1999a) is an event notification infrastructure that allows users to configure interest agents to observe the stream of events in the collaboration space.

Related Patterns

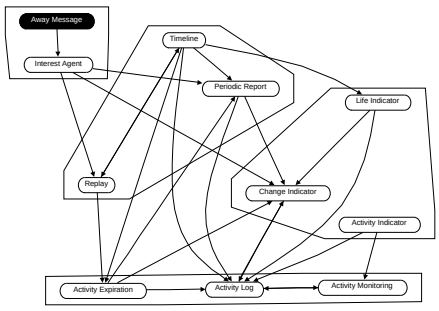
PERIODIC REPORT$_{\rightarrow 7}$**:** The notifications of the interest agent can be accumulated in a periodic report. This has the advantage that the owner of the agent will not be disturbed during the day.

CHANGE INDICATOR$_{\rightarrow 3}$ 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.

VISIBLE AUDIENCE$_{\rightarrow 10}$ 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 VISIBLE AUDIENCE pattern shows users who are currently following the group process (by being there).

REPLAY$_{\rightarrow 5}$ can be used by an INTEREST AGENT to provide a very detailed report of the activities.

9 Away Message



Alternative name(s): Auto reply

Intent Inform active users that a response to their request will be delayed.

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

Problem **Users expect that their interaction partners quickly respond to their actions. But sometimes, the interaction partner is unable to respond quickly. The longer the initiating users has to wait, the bigger is his disappointment.**

Scenario Martin has encountered problems to use the graphics components of the game engine. He thus contacts Paul and asks him a question. Normally, Paul responds after several minutes, but at the evening of Martin's working day, there is still no response.

Symptoms *You should 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 of their interaction partners based on previous experience. This means that they expect a specific responsiveness of their interaction partner.
- users stay away from the interaction space from time to time.

Solution **Therefore: Respond to an action of another user with an automatic away message whenever the normal response time cannot be guaranteed. Provide information when the requesting user can expect a response.**

Collaborations Basically, the Away Message pattern proposes to follow a three

step process when temporarily leaving an interaction context:

Setup: Before a user leaves the interaction space, he thinks about the time of absence. He creates an away message that explains why the user cannot respond and that includes information on the earliest possible reply.

In most cases, the user also provides an explanation how to handle urgent requests (e.g., by sending the message to a deputy).

Execution: When a sender sends a message to an absent receiver, the receiver's system automatically replies with the away message. To avoid duplicate notifications, the receiver's system in addition remembers that the sender was notified. Further mails of the sending user will not be automatically replied.

Tear Down: When the user returns, he deactivates the away message.

Rationale

The problem of responsiveness to activities has been studied by Tyler and Tang (2003) in the context of e-Mail communication. The authors performed a field study in a large technology company and interviewed employees about their e-mail usage habits. Of the 24 interviewed subjects, 17 used away messages frequently to signal absence. One user reported his expectations of away messages as follows: "You ask someone for information that they know, and you sit around waiting and waiting for them to get back to you, and you find out that they've been out of town.". The authors' analysis showed that one third of the subjects turned their away messages on if they were out of office for one day. One subject left it on even after returning from the trip until he managed to catch up with all messages sent to him.

The authors finally proposed an *expectation-to-breakdown timeline* that visualizes the perception of responsiveness. The basic idea is that users have an expectancy of the time that they wait for a reply. When they did not receive a reply after this threshold was passed, they will perceive a breakdown of collaboration. They will start to think about reshaping the collaboration in order to reach the goal. This can mean that they send a follow up message or that they try to reach the recipient using another media.

Figure 16 visualizes this understanding. The left part shows the normal behavior. A user sends a message (ore more general performs an activity) and the receiver processes the message and generates a reply. The middle part of Figure 16 shows the situation where the receiver is unable to create a reply. In this case, the sender will wait a specific time before he tries to remind the receiver that the sender is still expecting a reply. If the receiver still does not reply, the sender will probably try to reach the receiver using another communication channel. Each of these waiting times represent efforts at the sender's site.

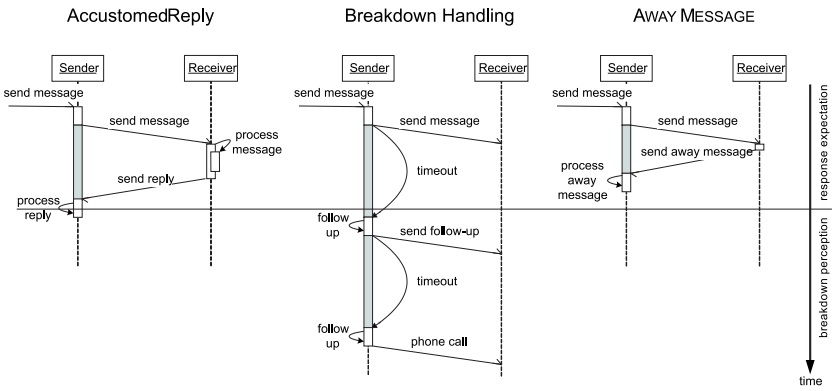


Figure 16: Communication patterns in a request-reply interaction.

The AWAY MESSAGE pattern reduces these waiting times as shown in the right part of Figure 16. Since the system automatically generates a reply, the sender can enter the breakdown handling immediately. The message provided by the receiver helps the sender to better understand the receiver's context and provides hints how to handle the breakdown.

Check

When applying this pattern, you should answer these questions:

- How do you let users configure their away message?
- How do you ensure that the away message contains all required information (e.g., the time when normal communication would reconvene)?
- Will users be informed about who has received an away message while they were absent?

Danger Spots

One of the largest problems with away messages is that they often do not distinguish between bilateral and group communication. When communicating by, e.g., a MAILING LIST$_{\rightarrow 10}$, the receiver's system may reply with an away message that is received by the whole group instead of the individual sender. The reason for that is that many mailing lists modify the e-mail headers so that the sender field is different to the reply-to field. The receiver's system keeps track of the senders but replies to the reply-to address. This address is again multiplexed to all members of the mailing list. Mailing list members may therefore receive multiple away messages, which is annoying.

A solution would be to keep track of the addresses to which an away message was sent instead of the senders to which the message was a reaction. An alternative solution that is often used is not to auto-reply an away message if the original message was not personally directed to the user. For e-mails this is, e.g., the case if the recipient is set to the address of the mailing list and the member of the mailing list is only added as a blind cc receiver to the message.

The groupware should ensure that the user switches the away message off when he is available again. You should also be aware of the problem that users might reply with a faked Away Message if they do not want to answer a message.

It is important that relevant context information is provided with the away message. A bad example of an away message can be found in the opening picture. The user stated that he would be back in 20 minutes. Some of the context information of this message is quite easy to reconstruct: Since the post-it is located at a door, one can assume that the owner of the room behind that door has placed the post-it there. But even this can be ambiguous if more people live behind that door. More difficult is the notion of time: Since the receiving user does not know when the absent user has created the away message, he will not be able to restore the context of the absent user and reconstruct the date information. A better message would name an exact date when the user will be back (e.g., the 17th of June 2007, 21:00h). In this case, no time context is required to understand the message.

Known Uses

Vacation (Costales 2002) is probably the most widely used implementation of an away message. In the activation phase, the users can specify a message body that is from then on sent to all users who send the receiver an e-mail.
Whenever replying with an away-message, the system keeps track of the sender and ensures that no duplicate messages are created.

```
Return-Path: <MAILER-DAEMON@mailstore.fernuni-hagen.de>
Received: from cl-mailhost.FernUni-Hagen.de ([132.176.114.188] verified)
  by mailstore.fernuni-hagen.de (CommuniGate Pro SMTP 5.0.2)
  with ESMTP id 15785207 for schuemm@mailstore.FernUni-Hagen.de; Sun, 07 May 2006 18:13:32 +0200

... Additional path headers ...

From: "Stephan Lukosch" <Stephan.Lukosch@FernUni-Hagen.de>
Date: Sun, 07 May 2006 18:13:32 +0200
Message-ID: <react-15785210@mailstore.fernuni-hagen.de>
X-Autogenerated: Reply
MIME-Version: 1.0
Content-Type: text/plain; charset="ISO-8859-1"
To: schuemm@mail.pi6.fernuni-hagen.de (Till Schuemmer)
Subject: Re: Committed new files -- please have a look
In-Reply-To: <445E1CAB.mail73B11SJ9K@afrika.pi6.fernuni-hagen.de>

I am out of the office until the 15th of May 2006. In urgent cases
please contact Simone Buecker (Simone.Buecker@FernUni-Hagen.de, +49
2331 987 4365). Your e-mail will not be forwarded.

Kind regards

Stephan Lukosch
```

Instant messaging systems like Trillian `http://www.trillian.cc/` allow the user to add a message that explains that the user is currently away. When another user initiates a chat communication, the system automatically replies with the provided away message. Note that Trillian only keeps track of the sessions to which an away message was sent. Messages are thus not sent twice in the same session but can be repeatedly sent to the same user in different sessions.

Related Patterns INTEREST AGENT$_{\rightarrow 8}$ is also configured by the absent user to act on his behalf. The difference is that the INTEREST AGENT keeps track of changes and therefore helps the user to stay informed instead of helping the group to stay aware of the absent user's absence.

AVAILABILITY STATUS$_{\rightarrow 10}$ also helps the senders to stay aware of the status of their request. However, the availability status normally does not reveal the temporal estimation of return.

10 Additional Thumbnails

Active Neighbors

Problem: The Local Awareness$_{\rightarrow 10}$ pattern only signals confocal users on the same artifact. If users work on related artifacts, they are not aware of each other, which implies that no collaboration will be established. On the other hand, especially collaboration on relate topics can support creative processes and mutual learning.

Solution: Provide awareness on peripheral activities that take place on related artifacts. Use a Semantic Distance$_{\rightarrow 10}$ to show how relevant those activity are. Rate activities on artifacts with a short semantic distance more important than activities with a long semantic distance. Ensure that activities on related artifacts do not distract the user's attention too much from the focused artifact.

Activity Counter

Problem: In a collection of shared objects, there may be more and less important objects. Especially for a newcomer there is no easy way to distinguish important from less important objects. This may result in a situation where the newcomer gets lost.

Solution: Add an activity counter to the visualization of the shared artifact. Artifacts that are important for the community will have a high number of activities such as visits, downloads, or updates. Unimportant artifacts may not attract as many visitors and therefore have a low activity counter value.

Attention Screen

Problem: Every request for attention needs to be processed by the user. Thus, it already takes some of his attention. But in situations, where the user needs to focus his attention on other things, this is disturbing.

Solution: Enable the user to filter the information which reaches him. Use meta-information (e.g. sender details) or content information (e.g. important keywords) to distinguish important information from unimportant information. Collect the less important information at a place where the user can process it on demand and forward relevant information directly to the user.

Availability Status

Problem: To allow spontaneous interaction, users have to be open for contact requests. But each request disturbs the contacted user or group in their current task. In addition, the importance of contact request may make it vital that the contact takes place.

Solution: Include an indicator in the application that signals the user's availability and how the user would react to an interaction.

Centralized Objects

Problem: To enable collaboration users must be able to share the data.

Solution: Manage the data necessary for collaboration on a server that is known to all users. Allow these users to access the data on the server.

Collaborative Session

Problem: Users need a shared context for their synchronous collaboration. But computer-mediated environments are neither concrete nor visible. This makes it difficult to define a shared context and thereby plan synchronous collaboration.

Solution: Model the context for synchronous collaboration as a shared session object. Visualize the session state and support users in starting, joining, leaving, and terminating the session. Automate the selection and start of tools.

Distributed Command

Problem: Clients can locally apply changes to replicated objects. When you distribute the new versions of locally changed replicated objects, you might distribute to much information than is necessary to keep the other replicas consistent. Especially, if only a small part a the replicated object has changed. This unnecessarily increases the network load and the response time of your application.

Solution: Encapsulate the users' changes as Commands (Gamma et al. 1995) and distribute the Commands via the network. Let other clients re-execute the Commands on their replica.

Embedded Chat

Problem: Users need to communicate. They are used to send electronic mail. But since e-mail is asynchronous by nature, it is often too slow to resolve issues that arise in synchronous collaboration.

Solution: Integrate a tool for quick synchronous interaction in your cooperative application. Let users send short text messages, distribute these messages to all other group members immediately, and display these messages at each group member's site.

Hall of Fame

Problem: Motivation for participation in a community is often related to the feedback that participants receive from the community. But often very active participants are not enough recognized by the community members.

Solution: Provide a list of those participants who participate most. Calculate the participants's participation level with respect to the degree that the participants helped others. Let each participant compare himself to those participants shown in the Hall of Fame.

Immutable Versions

Problem: Performing complex modifications on a shared object usually takes time and requires cognitive efforts of the user. If more users act on the same shared objects, the probability for conflicting changes increases. However, to rollback one of the conflicting changes is inappropriate since the user already spent too much efforts on performing the change.

Solution: Store copies of all artifacts in a version tree. Make sure that the versions stored in the version tree cannot be changed afterwards. Instead, allow users to store modifications of the version as new versions. Ask the users to merge parallel versions in the version tree unless they explicitly branch the version tree.

Intimacy Gradient

Problem: If users can always approach other users, the approached user may feel disturbed or even offended by the other user's interaction. Especially, they are drawn into an interaction that is not based on a mutual agreement to interact.

Solution: Arrange the interactions in an online community in a sequence which corresponds to their degrees of privateness. Require that users have succeeded in less private interaction before they can enter more private settings.

Local Awareness

Problem: Although most systems that work on shared data provide support for coordinating shared access, they often don't tell the user, who is working on a specific artifact. Such information is needed to establish ad-hoc teams that share a common focus. Without such information, users assume to work alone – and do not see the possibility or urge for collaboration.

Solution: Provide awareness in context. This means that the system tells the local user, who else is currently interested in the local user's focussed artifact and what they do with this artifact. Show this information whenever the artifact is shown on the screen. The information should contain details about the user drawn from his user profile, the artifact, and details on the activity,

which the user is performing. Ensure that the information is always valid.

Mailing List

Problem: Managing the set of recipients for group communication in a geographically distributed group is difficult and error-prone.

Solution: Establish a Mailing List for the group.

Masquerade

Problem: Your application monitors the local user. The gathered information is used to provide awareness information to remote users. While this is suitable in some situations, users often do not act as confident if they know that they are monitored. Users may feel the need of not providing any information to other users.

Solution: Let users control which information is revealed from their personal information in a specific interaction context. This means that the user should be able to filter the information which is revealed from his personal information. Remember to consider Reciprocity$_{\rightarrow 10}$.

Mediated Updates

Problem: Clients want to propagate update messages to other clients who keep replicas of the same data. If they contact the other clients directly, they have to maintain information who those clients are and have to establish communication with these clients. This is complicated and error-prone. Especially if some clients may disconnect and reconnect in an unpredictable way (if the set of clients changes over time).

Solution: After changing a replicated object inform a mediator which will distribute an update message to all interested clients.

Peer-to-Peer Update

Problem: Users change their local copies of the replicated artifacts and the other users cannot notice these local changes. This makes collaboration impossible.

Solution: After changing a replicated object locally send an update message for this object to all clients that also maintain a replica, take care that all clients receive this update message, and let these clients change their replica according to the information in the update message.

Personalized Attributes

Problem: When users work on shared data, they all have the same state. This implies that all views that are based on the shared model are showing the same content. Thus, the only way of interaction is tightly coupled interaction (WYSIWIS – What You See Is What I See). But there may be the need to relax this coupling so that users can for instance scroll to different screen regions (relaxed WYSIWIS).

Solution: Model application specific attributes of the shared data as personalized value holders that store the value for each user.

Presence Indicator

Problem: To provide awareness you connected other users' activities with artifacts which the local user focuses. But the surrounding of the artifacts provides only limited space for information. Awareness information thus competes with application data.

Solution: Limit the size of the awareness information's representation so that it uses only a small part of the available information channels. For a GUI system, this means that you should represent the confocal or peripheral users as a single icon instead of a long textual form. Focus on telling that there *are* other users, rather than providing much information on the other users' identity or task. Ensure that the indicator differs from the other artifacts representing application data.

Reciprocity

Problem: It is easy to agree on participation, if the goal is lucrative for everyone. But in many work situations, some people benefit more than others from a reached goal. This may frustrate active users.

Solution: Establish reciprocity. Ensure that all group members' activities result in an improved group result that is beneficial for all group members again. Prohibit people to benefit from group results if they are not willing to help the group in return.

Speed Replication

Problem: The response time of interactive applications has to be short. The network latency and delay wastes time in distributed systems. Thus interactive applications are inappropriate if the response time depends on client-server communication.

Solution: Replicate the shared data to the users' sites. Let a user change its local replicas and ensure consistency by using a Peer-to-Peer Update$_{\rightarrow 10}$.

Offline Replication

Problem: Users may not have a permanent connection to the system, where relevant data is kept. Without a permanent or just a poor connection to the data, users will not be able to finish their work, if the data cannot be accessed.

Solution: Replicate the data to the user's device. Update the replicas whenever two systems which hold copies of the data connect.

Semantic Distance

Problem: Your Semantic Net$_{\rightarrow 10}$ is very dense in a sense that artifacts have a semantic relation to many other artifacts. But not all artifacts have the same importance for the user. If the user sees only the semantic net, he might get lost in the diversity of relations.

Solution: Use weighted edges to describe the strength of the semantic relation. Interpret these edges as distances. If two artifacts are semantically strong related, ensure that the connecting edge in the Semantic Net represents a short distance.

Semantic Net

Problem: Detecting short semantic distances between artifacts based on a similarity measure often leads to ineffective and inexact results. It is time consuming, when there are many artifacts with large distances because this would involve much unnecessary computation. In addition it fails, if two artifacts are related by means of an intermediate artifact.

Solution: Produce a semantic net that contains artifacts and relations between artifacts. Relate two artifacts, if they have much in common (as in the Semantic Distance$_{\rightarrow 10}$ pattern). Define the distance between two artifacts as the length of the shortest path between these artifacts.

State Transfer

Problem: Users are collaborating in a Collaborative Session$_{\rightarrow 10}$ but not all users participate from the beginning. Due to this, these users do not know the intermediate results of the Collaborative Session$_{\rightarrow 10}$ which makes it difficult for them to collaborate.

Solution: Let latecomers ask an informed participant of the group to directly transfer the current state of a Collaborative Session$_{\rightarrow 10}$ to them. Ensure the consistency of the state.

User Gallery

Problem: If more than one user interacts with shared data, it is hard to coordinate the interaction - especially with strangers. Without knowing who is using the system, it is hard to establish collaboration or to become aware of other users' activities.

Solution: Provide a list of all users who are members of the community. Design this list in a way that it is interesting to browse.

Virtual Me

Problem: In a large user community, account names look similar. But users need to communicate their identity in order to interact with other users.

Solution: Allow the users to play theater! Provide them with means to create a virtual identity that represents them while they act in the system. Show the virtual identity when the user is active.

Visible Audience

Problem: Users are providing information for other users by means of shared objects. But making an object accessible does not ensure that the object was seen by other users.

Solution: Inform the author of a shared object when another user accesses this object.

References

Bentley, R., W. Appelt, U. Busbach, E. Hinrichs, D. Kerr, K. Sikkel, J. Trevor, and G. Woetzel (1997). 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*.

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

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

Costales, B. (2002). *sendmail* (3 ed.). O'Reilly.

Dourish, P. and V. Bellotti (1992). Awareness and coordination in shared workspaces. In *Conference proceedings on Computer-supported cooperative work*, pp. 107–114.

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

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

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

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

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

Ganoe, C. H., G. Convertino, and J. M. Carroll (2004). 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*, New York, NY, USA, pp. 453–454. ACM Press.

Ganoe, C. H., J. P. Somervell, D. C. Neale, P. L. Isenhour, J. M. Carroll, M. B. Rosson, and D. S. McCrickard (2003). 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*, New York, NY, USA, pp. 21–30. ACM Press.

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

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

Henkel, J. and A. Diwan (2005). Catchup!: capturing and replaying refactorings to support api evolution. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, New York, NY, USA, pp. 274–283. ACM Press.

Jackson, T. W., R. Dawson, and D. Wilson (2003). Understanding email interaction increases organizational productivity. *Commun. ACM 46*(8), 80–84.

Lukosch, S. (2003a, August). *Transparent and Flexible Data Sharing for Synchronous Groupware*. Schriften zu Kooperations- und Mediensystemen - Band 2. JOSEF EUL VERLAG GmbH, Lohmar - Köln.

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

Manohar, N. R. and A. Prakash (1995a, November). Dealing with synchronization and timing variability in the playback of session recordings. In *Proceedings of the Third ACM Multimedia Conference*, San Francisco, CA, USA, pp. 45–56.

Manohar, N. R. and A. Prakash (1995b, September). The session capture and replay paradigm for asynchronous collaboration. In *Proceedings of the Fourth European Conference on Computer Supported Cooperative Work*, Stockholm, Sweden, pp. 149–164.

Parsowith, S., G. Fitzpatrick, S. Kaplan, B. Segall, and J. Boot (1998). Tickertape: Notification and communication in a single line. Volume 00, Los Alamitos, CA, USA, pp. 139. IEEE Computer Society.

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

Schümmer, T. (2001). 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. IEEE-Press.

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

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

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

Schümmer, T. and S. Lukosch (2007). *Patterns for Computer-Mediated Interaction*. John Wiley and Sons Ltd. to appear.

Tidwell, J. (2006). *Designing Interfaces*. Sebastopol, CA, USA: O'Reilly.

Tyler, J. R. and J. C. Tang (2003, 14-18 September 2003). When can i expect an email response? a study of rhythms in email usage. In K. Kuutti, E. H. Karsten, G. Fitzpatrick, P. Dourish, and K. Schmidt (Eds.), *Proceedings of ECSCW2003*, Helsinki, Finland, pp. 239–258. Kluwer Academic Publishers.

Index