GAMA – A Pattern Language for Computer Supported Dynamic Collaboration

Till Schümmer
Computer Science VI - Distributed Systems
FernUniversitaet in Hagen
Informatikzentrum, Universitaetsstr. 1, D-58084 Hagen, Germany
till.schuemmer@fernuni-hagen.de

Abstract

The GAMA pattern language provides patterns for supporting dynamic teams with computer technology. In this document, a small subset of the GAMA language is presented that deals with spontaneous group formation based on common interests.

1 Introduction

New organizational models such as virtual organizations have made computer supported collaborative work (CSCW) more and more important (e.g. (Lillehagen, Dehli, Fjeld, Krogstie, and Jørgensen 2002)). One characteristic of virtual organizations is that the members of virtual organizations collaborate in order to reach a common goal. After the goal is reached, they may separate again. Thus organizations are compiled and reorganized on a frequent basis.

Members of virtual organizations have to accommodate the demand of frequent restructuring and fast team formation in their daily work. This is, why I refer to teams that work in such environments as *dynamic teams*. Some of the problems that arise for dynamic teams are

– how to find the expertise within the organization that is needed to fulfill an urgent task,

– how to find people who can collaborate on a specific task, or

– how to coordinate collaboration on shared documents.

Formal management is very difficult in environments of frequent restructuring – if not even impossible. In contrast or in addition to controlling such teams by formal management techniques, one should train the team members in several key practices: frequent communication for establishing mutual understanding, frequent collaboration to foster peer learning and knowledge transfer, and fast team formation to respond to ad hoc demands.

The focus of the GAMA pattern language presented in this paper is to provide design solutions to end-users and software developers that help to build CSCW technology, which supports the needs of teams in a frequently changing environment. It takes for granted that the team members have found a way to work on shared artifacts. This can, for instance, be the use of a shared file system, an editable web server (a WIKI (Leuf and Cunningham 2001)), or a version management tool. Another prerequisite is that basic communication technology is available (such as e-mail or chat systems).

This paper presents the part of the GAMA[1] language that addresses group formation. The basic idea (which is also expressed in the pattern FROM SHARED DATA TO SHARED WORK$_{\rightarrow 2.1}$) is that people notice one another, while they work on shared artifacts. They can then benefit from jointly working on the same or related shared artifacts through establishing a group collaborating on the shared artifact.

1.1 Instead of a disclaimer

There is one important limitation in the applicability of most patterns of the GAMA pattern language: If the organizational structure does not reward the process of knowledge sharing, all technical solutions will soon fail. This section presents one example, how technical solutions – although carefully designed – failed due to an inappropriate social environment.

It shows how a lack of such acceptance results in the failure of a knowledge management system is given by Pipek, Hinrichs, and Wulf (2003): They examined a German steel mill, where technical diagrams of the plant should be transferred to a central repository. But important parts of the knowledge about the steel mill's current state resided in the individual divisions. The central repository only held those diagrams, which were created in a planning phase of an installation. Ad-hoc changes were often not recorded in the central repository. Although the organization asked their members to provide such information, sharing did not take place.

This had two main reasons: first, the diagrams had to be filed by the archive group. This made changes in the repository difficult and time consuming. Second, the data stored in the repository was not as valuable as the data that could be maintained from the individual workers of each division. They knew their plant best – regardless the information, which was stored in the diagrams. And this knowledge was more than just knowledge. It made workers important and not exchangeable. It was a reason for other colleagues to contact them – and thus an important social capital. These were reasons for not sharing the knowledge in an automated way.

A conclusion that one can draw from this study is that it is often difficult to access shared data crossing different divisions of a company (or even different companies in a virtual organization). The factors outlined by Pipek et al. are only a few among many others see also Hinds and Pfeffer (2003).

[1]GAMA is an acronym that stands for group awareness based on multiple associations: initially, the language was heading for providing group awareness based on the relations between artifacts and users.

When reading and applying the following pattern language, it is important to get the focus of the desired collaboration straight. Is it collaboration between members of a small team who should collaborate on a specific project? Is it knowledge transfer between individuals within a large organization? Or is it collaboration within a virtual organization?

The answers to these questions will influence the access to shared data and the willingness to provide meta information on each individual's work. They also influence the willingness of collaboration between participants.

The following patterns are intended to be applied in a context where collaboration is desired and information can flow liberally. This might range from the full organization to a small team.

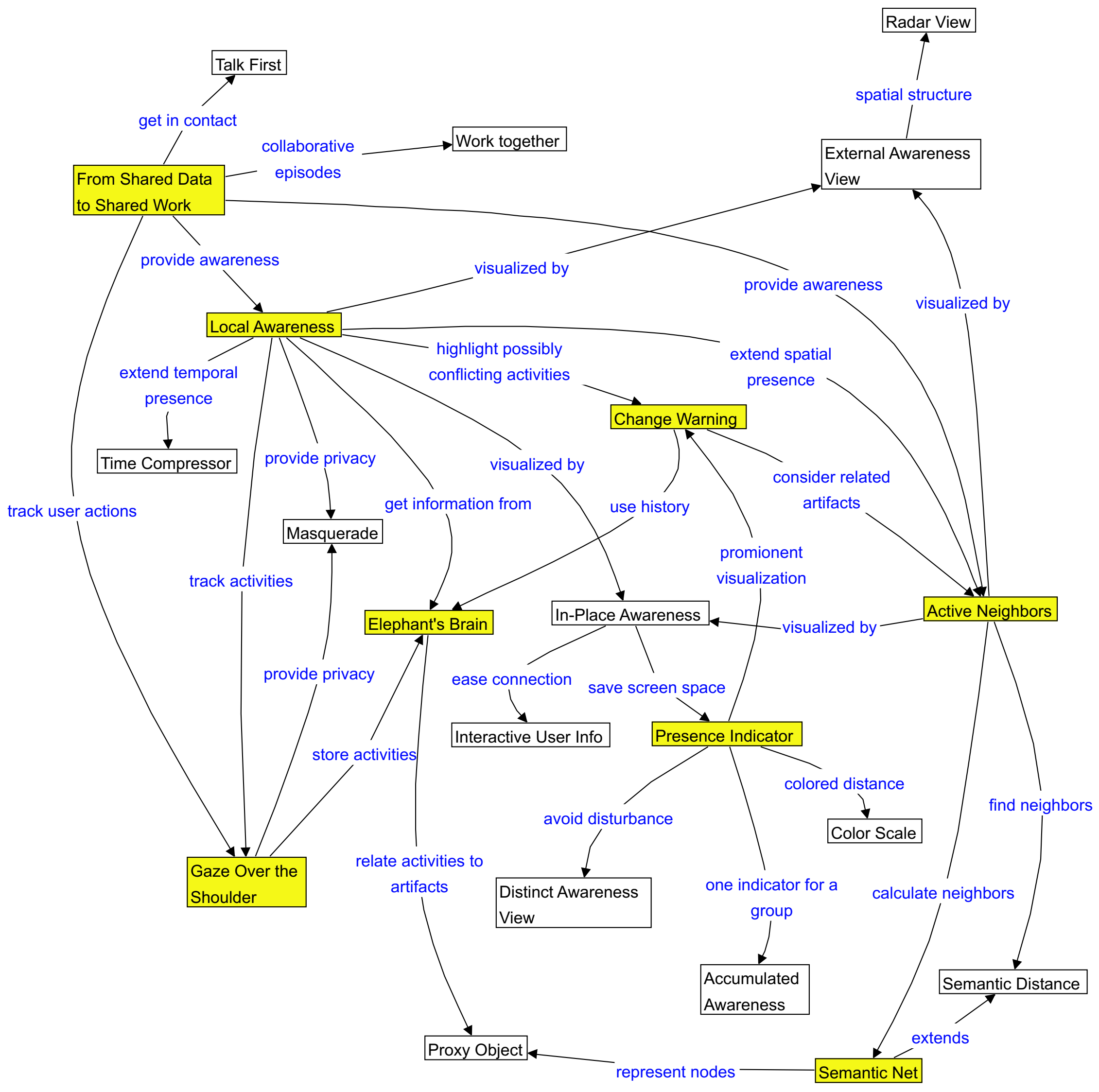


Figure 1: The structure of the GAMA pattern language.

1.2 How to use the GAMA pattern language

Figure 1 provides an overview of the full GAMA language, as it is planned. Patterns are represented as boxes. Some of the relations between the patterns are represented as directed links. The links propose different reasonable paths through the GAMA language. Note, figure 1 only shows a subset of the relations between patterns. One can refer to each pattern to find more relations to other patterns in the related patterns section. Each pattern in section 2 includes a miniature view of figure 1 with one black colored pattern. This visualization was inspired by Alexander, Ishikawa, and Silverstein (1968) and indicates the position of the current pattern in the pattern language.

This paper presents the following 8 complete patterns:

FROM SHARED DATA TO SHARED WORK$_{\rightarrow 2.1}$ addresses the problem of how to bring together people who could benefit from working with one another. It solves the problem by using other patterns of the GAMA language that provide awareness on users who work on a related artifact and propose collaboration.

LOCAL AWARENESS$_{\rightarrow 2.2}$ focusses on the situation, where several users work on the same artifact. It proposes to provide information on the other users in the current context of work (namely next to the artifact, which is currently used by the users).

PRESENCE INDICATOR$_{\rightarrow 2.3}$ solves the problem of how to visualize awareness information that is bound to an artifact. It shows a way, how this can be solved in a graphical user interface.

CHANGE WARNING$_{\rightarrow 2.4}$ is a pattern that provides awareness on activities, which modified the artifacts, in order to prevent conflicting work or work that is based on an obsolete perception of the artifacts.

ACTIVE NEIGHBORS$_{\rightarrow 2.5}$ extends the patterns LOCAL AWARENESS and CHANGE WARNING in a way that users are also aware of other users (or activities) that take (or took) place on semantically related artifacts, since these activities could affect the user's work. It uses a SEMANTIC NET to calculate users who could benefit from collaboration.

GAZE OVER THE SHOULDER$_{\rightarrow 2.6}$ provides the solution to technical, economic, or legal obstacles for the use of the awareness patterns above: it is often not possible to change the current application that is used to work on the artifacts. The solution is to monitor external communication channels to maintain and process additional information on users' activities with the artifacts.

ELEPHANT'S BRAIN$_{\rightarrow 2.7}$ provides a solution for remembering activities that were performed by the users. Such information is needed by the awareness patterns as a basis for the calculation of users or activities that other users should be aware of.

SEMANTIC NET$_{\rightarrow 2.8}$ finally shows how one can calculate the semantic distance of artifacts. It solves this question by using a graph data structure and applying graph algorithms instead of calculating distances between every two artifacts.

Theses patterns provide the basic functionality for providing group awareness with the GAMA language. Section 3 lists the intents of the remaining 15 patterns of the GAMA pattern language. These patterns are subject to further development.

1.3 The Pattern Structure

Each pattern is presented in an Alexandrian form. The pattern name appears as a section title followed by other possible names for the pattern (AKA), the intent, and the context of the pattern. It helps the reader to decide, whether or not the following pattern may fit into his current situation.

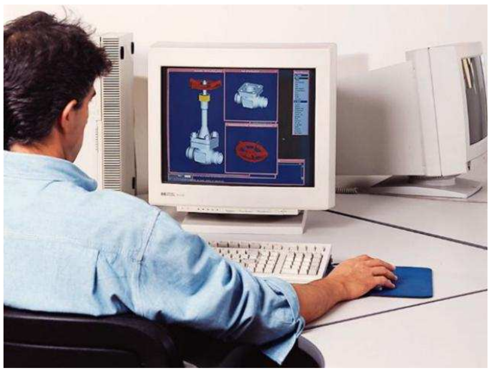
Then follows the core of the pattern composed of the problem and the solution statement in bold font separated by a scenario and a symptoms section. The scenario is a concrete description of a situation where the pattern could be used, which makes the tension of the problem statement (the conflicting forces) tangible. The symptoms section helps to identify the need for the pattern by describing aspects of the situation more abstract again. It lists observable forces that are unbalanced before the pattern was applied.

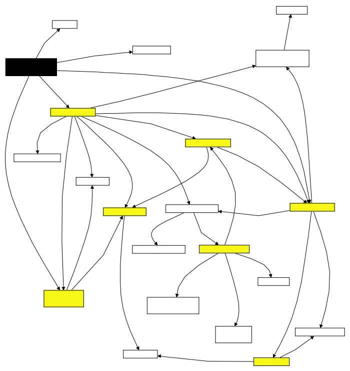
After the solution section, the solution is explained in more detail (participants, rationale, danger spots, known uses) and indications for further improvement after applying the pattern are provided (in the related patterns section). The participants section explains the main components or actors that interact in the pattern and explains how they relate to each other. The rationale section explains, why the forces are resolved by the pattern. Unfortunately, the application of a pattern can in some cases raise new unbalanced forces. These counter forces are described in the section labelled Danger Spots.

References to patterns are shown in SMALL CAPS. If the pattern is part of the GAMA pattern language, the section in which the pattern is explained is provided behind the pattern.

2 The GAMA Pattern Language

2.1 From Shared Data to Shared Work





Intent

Foster group formation and spontaneous collaboration by bringing together users who share an interest in the same data and providing them with means for communication and collaboration.

The pattern is an overview pattern, which sets the stage for the remaining patterns of the GAMA language.

Context

Users are working with an application that provides access to shared data. They consume or manipulate the shared data using their personal clients, which may but don't have to be at diverse locations.

The shared data may be, for instance, a file, a set of files that are used for a project, a cluster of related objects in a distributed application, or a set of records in a database that is accessed by all users.

The interaction with the shared data is not easily predictable, since the possible activities do not follow a fixed set of workflows.

——— ◊◊◊ ———

Problem

Although many users work with the same shared data, they may not recognize the other users' work. This results in parallel or conflicting work and a lack of collaboration and learning from one another.

Scenario

Imagine a scenario in the automobile industry: Many employees work on the design of new car. Some experts design the electronics, while others work on the layout of the control panel. In each group there are different responsibilities and roles. While the designer arranges the tachometer, the ecologist considers how the tachometer may be recycled. All engaged employees work on the same shared data – namely the design documents of the whole car.

During the design process different modes of collaboration (Schümmer and Haake 2001) may take place: First the designer works on his own to create a first draft. If the material for the tachometer is to be specified, the situation demands a close cooperation with the ecologist. But the ecologist is not the only person the designer needs to collaborate with. Together with a mechanic, he has to investigate, whether or not the tachometer can be created with the considered material. These are only two examples of collaboration on shared artifacts, which is needed in a complex design process.

The more complex and dynamic the environment (respectively the shared data) gets, the harder it is to predict all valuable constellations for collaboration by means of a strict work flow.

Consider as an less obvious situation in the car example the design of safety seats for babies and the development of an airbag security system. In most modern cars that have a co-driver airbag, one can no longer place the safety seat for babies on this seat since the inflation of the airbag may kill the baby sitting in the safety seat. In this example, two designers did not collaborate although they worked with the same space in the car. The result is that some requirements were not communicated among the two designers.

The examples show that collaboration opportunities are often not recognized and collaboration does not take place. This may have two consequences. The most obvious consequence is that the work of the different participants will result in conflicts. If the designer does not contact the electrical engineer and the mechanical engineer, it might be that the design of the tachometer does not provide any space for the needed cables that lead the signals to the device. In this case, the designer obviously didn't know much about the technical constraints of his design, which leads to the second consequence: The designer would not be able to learn from the electrical engineer. If they had collaborated, they could have discussed their different experiences and won new insights.

From this scenario, one can see that it can be very important to foster collaboration, when two or more users work on the same part of the shared data.

Symptoms

The problem becomes obvious when ...

- managers consider it as difficult to predict who should collaborate to reach a specific goal.
- team members notice that they worked on the same artifacts after the work is finished.
- parallel work causes conflicts that could have been resolved when the team members had worked together.

Users are unaware of the potential for collaborative activities that could improve the overall result or minimize conflicting work.

The users are working with the shared data without much knowledge of their colleagues (fig. 2). The only effect that let's them deduce that other users are working on the shared data is

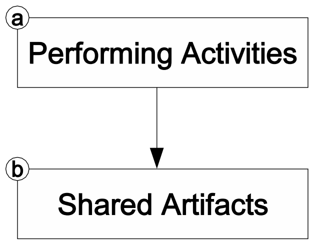


Figure 2: Single user's work on shared data.

that the data changes from time to time. Uncoordinated collaboration in a shared filesystem is an example of this kind of work.

One of the forces in the context section stated that it is difficult to know all useful points of collaboration in advance. But even if this knowledge exists, it is still challenging to establish groups. Users have to coordinate collaborative episodes before they actually take place (cf. fig. 3).

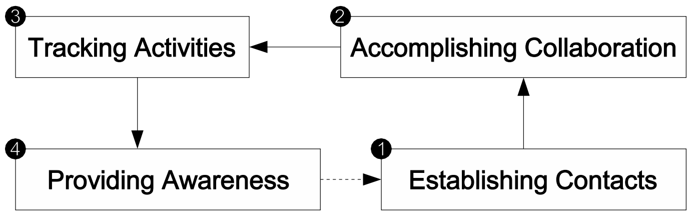


Figure 3: User driven group formation.

First, the users have to identify other users who should be part of the team and contact them to propose a collaborative session (cf. fig. 3-1). GAMA refers to this kind of group formation as *user driven group formation.* If all participants agree, they start a collaborative session (2). The topic of the session sets a new focus – the group focus – that all participants need to share. Establishing group focus may be difficult if the group members were focussing on unrelated activities when the group started its session.

Depending on the desired strength of collaboration, group members need to be aware of each other's activities during the collaborative session. The users' activities need to be tracked (3) to provide awareness (4). Awareness information is mainly used to coordinate the activities in the collaborative session and to avoid conflicting activities in the same workspace.

Solution

Therefore: Model dynamic groups that are built from the set of users of the shared data who share a common interest (e.g. detected by patterns LOCAL AWARENESS$_{\rightarrow 2.2}$ **or** CHANGE WARNING$_{\rightarrow 2.4}$**).**

Allow these groups to be established on the basis of their members' activities (again LOCAL AWARENESS$_{\rightarrow 2.2}$), so that they are likely to have a common interest. Provide them with collaborative tools that support communication (TALK FIRST$_{\rightarrow 3}$) and collaboration (implementing various groupware patterns).

Use the patterns of the related patterns section to detail each of the actions proposed by the above paragraphs.

◊◊◊

Participants

User: The user works with an application on shared data. His work initially takes place in a single user mode. After becoming aware of other users, he starts a group with these users to enter closer collaboration.

Activity: Activities are objectifications of the actions the user performs. They represent a semantic entity formed of low-level interactions, such as key-strokes or mouse movements. Examples for activities could be

- navigating from one web page to another,
- performing a search query,
- working in the same shared space (and in this case interpreting the content of the shared space as shared data), or
- changing a method of a source file in a programming environment.

As the examples suggest, activities should be at a semantic level, which make sense for the end user.

Artifact: A set of artifacts represents the shared data. Activities focus on artifacts.

Group: When users decide to collaborate, they form a group. This group then interacts using shared tools.

The participants (User, Activity, Artifact, and Group) collaborate in three main use cases that are addressed by related patterns:

Tracking Activities: Users perform activities on artifacts, which have to be tracked to be able to provide awareness (GAZE OVER THE SHOULDER$_{\rightarrow 2.6}$).

Providing Awareness: Based on tracked activities, the system calculates awareness information, which is indicated to users (LOCAL AWARENESS$_{\rightarrow 2.2}$, CHANGE WARNING$_{\rightarrow 2.4}$).

Establishing Contacts: Users perceive awareness information. This means that they know, which other users they can contact to form a group (TALK FIRST$_{\rightarrow 3}$).

Rationale

The pattern FROM SHARED DATA TO SHARED WORK emphasizes the importance of providing group awareness (cf. fig. 4). The group formation process starts in the phase of single-user work. While the users work independently (a) on shared artifacts (b), the system tracks their activities (1). Based on this information, the system can provide awareness information about the users' single-user activities (2). This information can, for instance, reveal whether other users currently work on the same artifacts. If a user detects another user who works in the same area, she can establish a contact with this user (3). I call this kind of group building *data and activity driven group formation.* Compared to the user driven group formation, data and activity driven group formation ensures that the users share a common focus, which makes focussing the group and accomplishing collaboration (4) much easier. In this model, awareness information is an important means to detect collaboration opportunities. Note that it also plays a role within tightly coupled collaboration, where awareness information can be implemented in the same way, as if the group formation had been user driven.

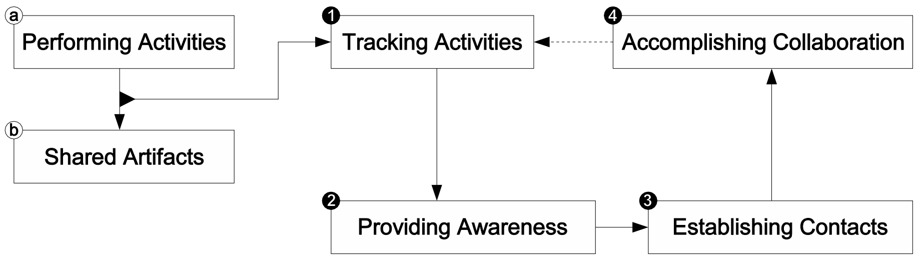


Figure 4: Data and activity driven group formation.

One might ask, how this solution differs from the application of a traditional version management system, like CVS (Price 2000) or Envy (Pelrine, Knight, and Cho 2001). Version management systems also provide means for coordinating parallel work on shared data. If two or more users work at the same time with the same artifact, most version management systems create parallel editions of the artifact and leave the integration of the two editions to the end user. That actually is the benefit that would make a user asking for the pattern FROM SHARED DATA TO SHARED WORK: Knowing in advance, who works with an artifact, can make the user more cautious because he knows that all work that he does will conflict with other users' work. Collaboration at this point in time combines the integration aspect with the actual work – and puts the efforts for integration on the shoulders of all co-workers.

Danger Spots Providing mechanisms for dynamic artifact-based group formation may lead to a practice that disregards formal team management at all. The pattern does not intend that formal work flows should be discarded, but that they need to be accompanied by dynamic group formation.

When designing the systems, you should take into account that users are often reluctant to change their current work environment.

Thus, it is likely that totally new applications will not be used only for the sake of better group awareness (cf. e.g. (Sohlenkamp, Prinz, and Fuchs 2000)). You should integrate the group awareness and team formation mechanisms as tight as possible. The ideal case would not require any user actions for setting up the service, and users would only notice the changed application, when collaboration opportunities arose.

Known Uses

TUKAN (Schümmer and Schümmer 2001) is a collaborative programming environment, which supports dynamic groups. It proposes a lightweight software development process. The developers first agree on a set of tasks for the construction of the software system. These tasks are written on shared planning cards (a distributed variant of the planning game, as it is proposed in *eXtreme Programming* (Beck 1999)).

A typical development cycle starts in a single user mode (or with two developers working together at one machine). The developer selects an appropriate task and starts coding, as if he was working alone. At the same time, other developers also start their work. All parties ignore one another until they become aware of each other because they are browsing or changing semantically related methods. The developers use the built-in communication support to get in contact with each other and finally decide that they should solve the part of the task together, which affects both of them. The system provides means for synchronous collaboration such as a collaborative code editor.

TUKAN was the main application, from which this pattern language was mined. Thus, it will be used as example in many of the following patterns.

GAMA Mall (Schümmer 2002) is a design study of a collaborative shopping application, which extends the web pages of the Amazon.com™ bookstore. Users become aware of other users browsing the store. The system informs the users if other users browse content, which is semantically relevant to the local user. Users can then initiate tighter collaboration, such as chatting or collaborative browsing.

I2I (Bradshaw, Budzik, Fu, and Hammond 2002) is a system that provides awareness on other users' activities on the web. Whenever two or more users browse the same or a semantically related artifact, the system visualizes this as a PRESENCE INDICATOR$_{\rightarrow 2.3}$. Users are provided with means for communication to establish contacts to users working on related artifacts (TALK FIRST$_{\rightarrow 3}$).

Related Patterns Related patterns are grouped into four sections: patterns for providing awareness, patterns for tracking activities, patterns for establishing contacts, and patterns for accomplishing collaboration.

You should first look at the pattern LOCAL AWARENESS$_{\rightarrow 2.2}$. This pattern outlines the most basic way of providing awareness to users of a system with shared data. It can be refined by extending the notion of locality. The pattern ACTIVE NEIGHBORS$_{\rightarrow 2.5}$ provides a starting point for this.

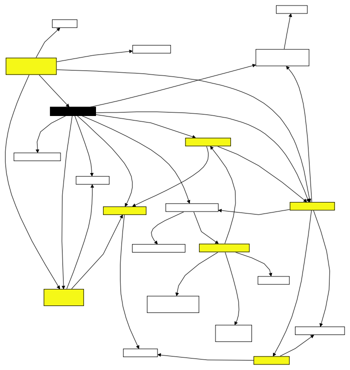
All patterns that aim on the provision of awareness rely on other patterns, which track the required data from the user actions. The starting pattern for this section is GAZE OVER THE SHOULDER$_{\rightarrow 2.6}$.

If you feel that the application provides enough awareness, you can figure out how contact is established (cf. TALK FIRST$_{\rightarrow 3}$) and which means you may use to support collaborative episodes (cf. WORK TOGETHER$_{\rightarrow 3}$).

Note that the area of supporting collaborative episodes would naturally comprise a whole pattern language for groupware applications. This would go beyond the scope of the GAMA pattern language presented here.

The Pattern ADHOC MEETING (Coldeway 2003) describes, how people should meet dynamically whenever an important issue arises. Although it does not relate to any technical issues or distributed team settings, it shares the same idea.

2.2 Local Awareness



AKA Contextual Awareness, Document Awareness (e.g. (Cohen, Jacovi, Maarek, and Soroka 2000))

Intent Provide awareness information in the context of the artifact, which is in the local user's focus.

Context Many users are working on a set of shared artifacts. You have already decided to use the pattern From Shared Data to Shared Work$_{\rightarrow 2.1}$ to encourage the formation of dynamic teams. The groups should be focussed on a shared current interest, which matches their current actions.

The application, which is used to access the shared data only provides means for manipulating or accessing the data and for keeping it consistent.

Users' current state is not yet reflected in the user interface. Especially, there is no means to find out whether or not other users are available who share a common interest.

◇◇◇

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.**

Scenario Imagine a real world plaza, where people meet for social interaction. This plaza is popular, since it offers many services, such as cafés to hang around, notice boards where citizens of the community post their private adverts, and shops where one can buy, for instance, food or books. Since the plaza is crowded, it motivates for talking or resting in the community.

Now consider a virtual community web site. It also offers bulletin boards, chat-rooms, and links to virtual shops. All these facilities are arranged on a central portal page. Although all communication means and services are available, the users often feel isolated at the portal page. The community web site looks as if no users were using it and does not motivate people to rest and talk in the community. Especially, the users don't see a way to meet by chance, which is one characteristic of a real-world plaza.

Symptoms

The problem becomes obvious when ...

- users cannot say, whether or not other users work with the same artifacts.
- users always have the impression of being on their own.

The background of the problem lies in the different nature of real-world interaction and virtual interaction. If people interact with real-world-artifacts, they have to be physically co-located with this artifact. In the case where more than one user works with the artifact, these users have to be co-located as well since all of them have to be co-located with the artifact. When users interact with artifacts in a virtual space, co-presence is not needed since all interaction is computer-mediated (and thus can be performed by means of computer networks). The users still interact with and have a relation to the artifact, but they don't have to get in contact with the other users anymore.

Solution

Therefore: 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.

——— ◇◇◇ ———

Participants

Artifact: The unit of shared data, where the users work on. The granularity of the artifacts has to be determined by the application designer. Examples are web sites, single pages of a web site, or more application specific artifacts, such as classes and methods in a programming environment.

Activity: The representation of a user's interaction with the shared artifact. It may be a modifying activity such as changing a web page or a reading activity, where the artifact is perceived by the user. In many contexts, it is not trivial to detect, which activities take place. The GAZE OVER THE SHOULDER$_{\rightarrow 2.6}$ pattern points out how activities can be detected.

Local User / Confocal Users: The local user interacts with the system and wants to get informed about confocal users who interact with the system from a distant location. *Two or more users are considered as* ***confocal****, if their current focus is on* *the same artifact.* This term was chosen to clearly distinguish between physically co-located users who work at the same physical location and confocal users who work with the same virtual artifact.

The set of confocal users on an artifact contains all users who have started an activity with the artifact, which is not yet finished. Several known uses don't distinguish between local and remote users when looking for activities. This implies that the set of confocal users for a specific artifact that is currently watched by the local user always includes the local user. Note that all users are local users at their machine. Making the distinction on the other hand reduces awareness information to awareness on remote users (only activities performed by remote users are considered).

Awareness Information: Awareness information represents the status of remote users and is displayed to the local user. The information may be at different levels of detail. For a small set of confocal users, it may provide information on the other user's identity.

Rationale

By explicitly telling the user that also other users are working with the artifacts (or more general at the same virtual location), these users get aware of each other, which is the basis for establishing contacts.

As with FROM SHARED DATA TO SHARED WORK$_{\rightarrow 2.1}$, one might ask how this solution differs from the application of a traditional version management system. If the only goal would be to avoid other users from overwriting their colleagues work, this might well be reached by using a version management system, which avoids parallel work on the same artifact (locking) or highlights conflicts before they can be committed to the version management system (such as in CVS (Price 2000)).

But the important point in this pattern is that the user gets aware of activities that currently take place on the artifact – not activities that are completed as it would be the case, when a user decided to check in a new version. This opens the opportunity to collaborate *within the activity* or adapt the activity to the changed situation as soon as possible.

Another difference to version management systems is that the LOCAL AWARENESS pattern considers a broader range of activities – not only modifying accesses like in version management systems but also reading accesses to the shared data.

Danger Spots

The granularity of artifacts and the size of the user community determines, whether or not users will be aware of one another.

If the system is very crowded (i.e. many users are working around a small set of artifacts), then the number of confocal users will be large at each point in time. Displaying all confocal users as distinguishable persons may produce an information overload and distract the local user from his task. In this case, it is better to simply tell that there are some or a specific number of confocal users without revealing their identity. On the other hand, displaying only the information that users are sharing a focus on the artifact without providing numbers for the size of the crowd does not provide any useful information in this case. This indicator will always be active.

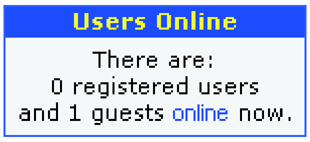
If the system provides access to many artifacts and the group of users is small, the probability that two users work on the same artifact will be very small. In this case, you can either reduce the granularity of artifacts or apply the pattern ACTIVE NEIGHBORS$_{\rightarrow 2.5}$, which extends the presence to other semantically related artifacts.

An important issue for the acceptance of the pattern is trust and privacy. User monitoring does only work, if it is mutually accepted by all participants. Otherwise, you will soon get the effect that users complain about being monitored or don't use the awareness features anymore. The pattern MASQUERADE$_{\rightarrow 3}$ addresses this problem.

When using this pattern, you should not mix up tasks and activities: This pattern does only propagate current foci of users in terms of the artifact that they are currently interested in. It does not reveal information on the users' current task besides the interaction that can be monitored from the activity itself. The question that this pattern solves is not how to bring together two or more people with a related task, but how to bring together people who use the same artifact to fulfill their potentially independent tasks.

Known Uses

Textual Indication on Community Websites Many community web sites tell, how many confocal users are currently on the site. The notion of focus is very broad at most of the sites. A user can have a focus on the site, but not on individual pages. An example for this kind of LOCAL AWARENESS is the home page of phpWebThings, an open source portal system (http://www.webservicesnet.com/wt/news.php). It shows, how many other users are currently connected to the site as a text message (fig. 5).

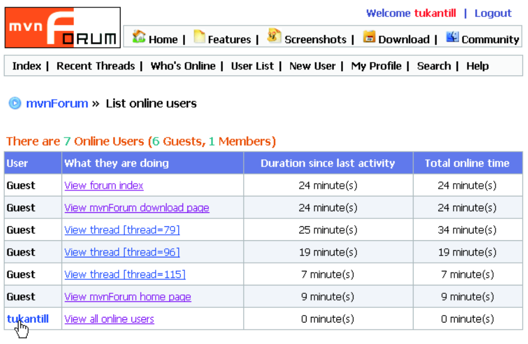

Users Online

There are:
0 registered users
and 1 guests online now.

Figure 5: Confocal users at the phpWebThings community site.

This kind of awareness information is the easiest variant of the pattern. All users are interacting with the same artifact (namely the web site). It is not distinguished between local and remote users. Users are able to surf the web site anonymously as guests but not without leaving traces.

Some indicators at web sites extend the awareness information to tell on which page the other users currently are. An example for this is the community page www.mvnforum.com (fig. 6). It shows the duration that they spent on the page and how long they are on the site besides the users' current focus (the page they are browsing).


Welcome tukantill | Logout
mvn FORUM
Home | Features | Screenshots | Download | Community
Index | Recent Threads | Who's Online | User List | New User | My Profile | Search | Help
mvnForum » List online users

There are 7 Online Users (6 Guests, 1 Members)

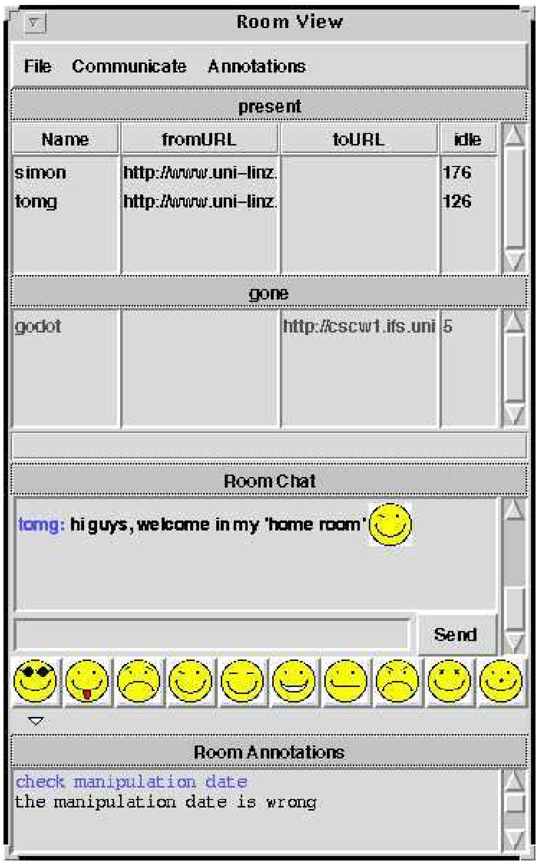
User	What they are doing	Duration since last activity	Total online time
Guest	View forum index	24 minute(s)	24 minute(s)
Guest	View mvnForum download page	24 minute(s)	24 minute(s)
Guest	View thread [thread=79]	25 minute(s)	34 minute(s)
Guest	View thread [thread=96]	19 minute(s)	19 minute(s)
Guest	View thread [thread=115]	7 minute(s)	7 minute(s)
Guest	View mvnForum home page	9 minute(s)	9 minute(s)
tukantill	View all online users	0 minute(s)	0 minute(s)

Figure 6: Confocal users at the mvnForum community site (http://www.mvnforum.com/mvnforum/listonlineusers).

CSCW3 The CSCW3 prototype (Gross 1999) is a system that shows for a given URL, who else is browsing this URL. It is a special kind of browser that logs user activities and displays confocal users in an external window (fig. 7). The local user can see, who is currently on the page and who has been on the page recently. The latter is a behavior that goes beyond the pattern's proposed functionality. It is described in the TIME COMPRESSOR$_{\rightarrow 3}$ pattern.

Odigo (Odigo 2001) is an instant messaging client, which tracks the users while they are navigating the WWW. It shows confocal users in an external view (fig. 8). The user can chose, whether he is interested in the current page or in the whole web site.

CoCoBrowse (Ter Hofte, Otte, and van der Gaast 1997) is a specialized browser, which shows in an attached window who else is currently viewing or editing a specific web page.



Name	fromURL	toURL	idle
simon	http://www.uni-linz.		176
tomg	http://www.uni-linz.		126

godot		http://cscw1.ifs.uni	5

tomg: hi guys, welcome in my 'home room'

check manipulation date
the manipulation date is wrong

Figure 7: The CSCW3 prototype (Gross 1999).

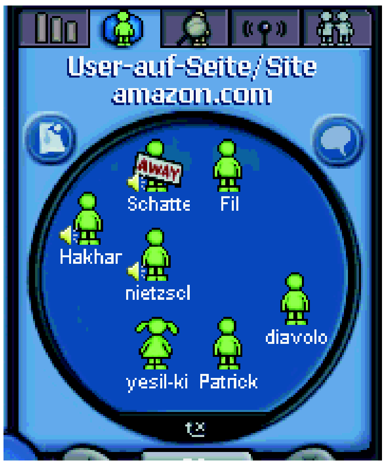


Figure 8: Odigo (Odigo 2001) showing confocal users at www.amazon.com.

Related Patterns GAZE OVER THE SHOULDER$_{\rightarrow 2.6}$ tracks user activities. Information on users' activities is needed to calculate the set of confocal users.

TIME COMPRESSOR$_{\rightarrow 3}$: Use the time compressor pattern, when users don't work at the same artifact at the same time and you still want to bring together users who worked on the same artifact successively.

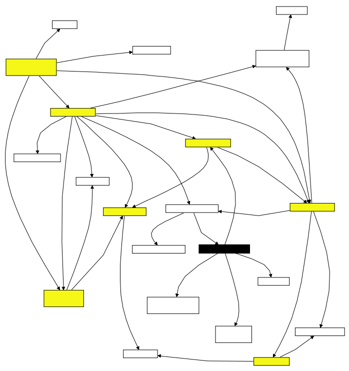
PRESENCE INDICATOR$_{\rightarrow 2.3}$: If screen space is limited, you can use a PRESENCE INDICATOR$_{\rightarrow 2.3}$ to present information on the confocal users in an iconic representation that saves space.

CHANGE WARNING$_{\rightarrow 2.4}$: For simplicity reasons, the current pattern assumed that all activities have the same quality. This is not the case for systems, where users can change artifacts. In these systems, you should consider to highlight possibly conflicting activities using the CHANGE WARNING$_{\rightarrow 2.4}$.

USER LISTS have been applied frequently in shared workspace systems. Although these lists have many commonalities (the visualization can, for instance, be the same), there is a crucial difference in the intent. Traditional USER LISTS intend to provide awareness on participants who have already joined a collaborative session. Thus, group formation is not a goal of USER LISTS.

PUBLISHER-SUBSCRIBER (Buschmann, Meunier, Rohnert, Sommerlad, and Stal 1996) can be used to can keep the calculation up-to-date.

2.3 Presence Indicator



Intent

Indicate that remote users look at an artifact, which is visible on the local user's screen.

Context

You have generated awareness information on confocal users using the Local Awareness$_{\rightarrow 2.2}$ pattern or on peripheral users with the Active Neighbors$_{\rightarrow 2.5}$ pattern. You have also decided to use an In-Place Awareness View$_{\rightarrow 3}$ to ease the process of connecting awareness information with those artifacts that are affected by the awareness information. This means that you are showing the awareness information in a close proximity to the original artifact's visualization.

Now, you are thinking about how to visualize the awareness information gathered.

◇◇◇

Problem

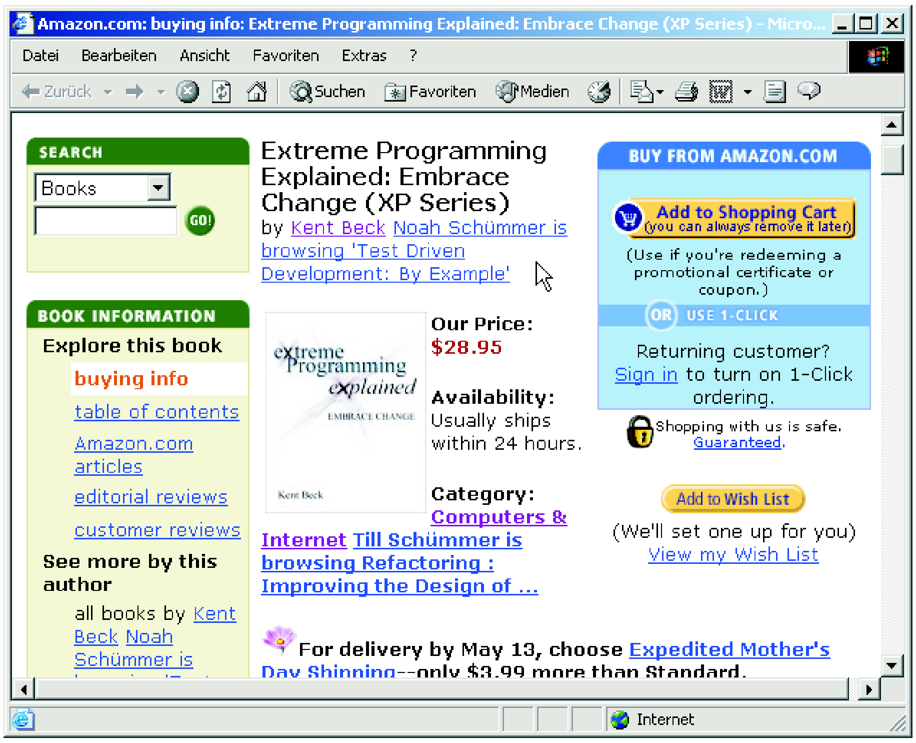
The In-Place Awareness View$_{\rightarrow 3}$ makes it easy to connect other users' activities with focussed artifacts. But the surrounding of the artifact only provides limited space for information. Awareness information thus competes with application data.

Scenario

Think of several users who are looking for books in a virtual bookstore. They want to be aware of one another to discuss the books, if they are interested in the same book. The virtual bookstore provides Local Awareness$_{\rightarrow 2.2}$ and Active Neighbors$_{\rightarrow 2.5}$ to calculate confocal users or users who work in a semantic nearness of the displayed artifacts.

Figure 9 shows how awareness information was displayed on the screen. The developers chose to provide the awareness information next to the artifact's representation on the screen. This information can be found next to the mouse pointer in figure 9.

The explanations of the awareness information clutters the whole page layout because it uses too many words. In addition, the textual representation of the awareness information is not easily distinguishable from the domain information (in this case the information on books). That makes the information hard to recept


Amazon.com: buying info: Extreme Programming Explained: Embrace Change (XP Series) - Micro...

Datei Bearbeiten Ansicht Favoriten Extras ?

Zurück Suchen Favoriten Medien

SEARCH

Books

GO!

BOOK INFORMATION

Explore this book

buying info

table of contents

Amazon.com articles

editorial reviews

customer reviews

See more by this author

all books by Kent Beck Noah Schümmer is

Extreme Programming Explained: Embrace Change (XP Series)

by Kent Beck Noah Schümmer is browsing 'Test Driven Development: By Example'

Our Price: $28.95

Availability: Usually ships within 24 hours.

Category: Computers & Internet Till Schümmer is browsing Refactoring : Improving the Design of ...

BUY FROM AMAZON.COM

Add to Shopping Cart (you can always remove it later)

(Use if you're redeeming a promotional certificate or coupon.)

OR USE 1-CLICK

Returning customer? Sign in to turn on 1-Click ordering.

Shopping with us is safe. Guaranteed.

Add to Wish List

(We'll set one up for you)

View my Wish List

For delivery by May 13, choose Expedited Mother's Day Shipping--only $3.99 more than Standard.

Internet

Figure 9: Textual in-place visualization of confocal users in a virtual bookstore.

and disturbs the user from his actual task (to look for books).

Symptoms

The problem becomes obvious when ...

– the available information channels to transmit information to the user don't have much more capacity, than which is needed to transmit the shared data.

– in GUI systems, there is not much space for showing information next to the artifact so that the awareness information destroys the screen layout.

– external awareness views, such as a Radar View$_{\rightarrow 3}$ do not work, because the artifacts cannot easily be described without their original context. For that reason, users find it hard to associate awareness information with the corresponding focussed artifact.

Solution

Therefore: 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.

◊◊◊

Participants

The main participant of the pattern is the *indicator*, which represents the awareness information. Each indicator displays awareness information for activities of confocal users or active neighbors that are referencing one specific artifact. The indicator is placed right next to this artifact. The indicator can be an icon in GUI systems, a special sound in audible interfaces, or any real-world object in *tangible interfaces*. It should be distinctive from the application's shared data.

Wherever possible, the indicator has built-in behaviour, which reveals more clues on the awareness information. For instance, if *tool tips* are available in the user interface, the indicator can explain itself with a tool tip when the mouse pointer touches it.

The sets of *confocal users* or *active neighbors* can be characterized by different group aspects:

– the cardinality (i.e. the number of confocal or nearby users),
– the distance to the artifact (this is always 0 if the users are confocal; for active neighbors, it represents the semantic distance of the user to the artifact), and
– the users' identities (which is uncomplicated if it is only one confocal user).

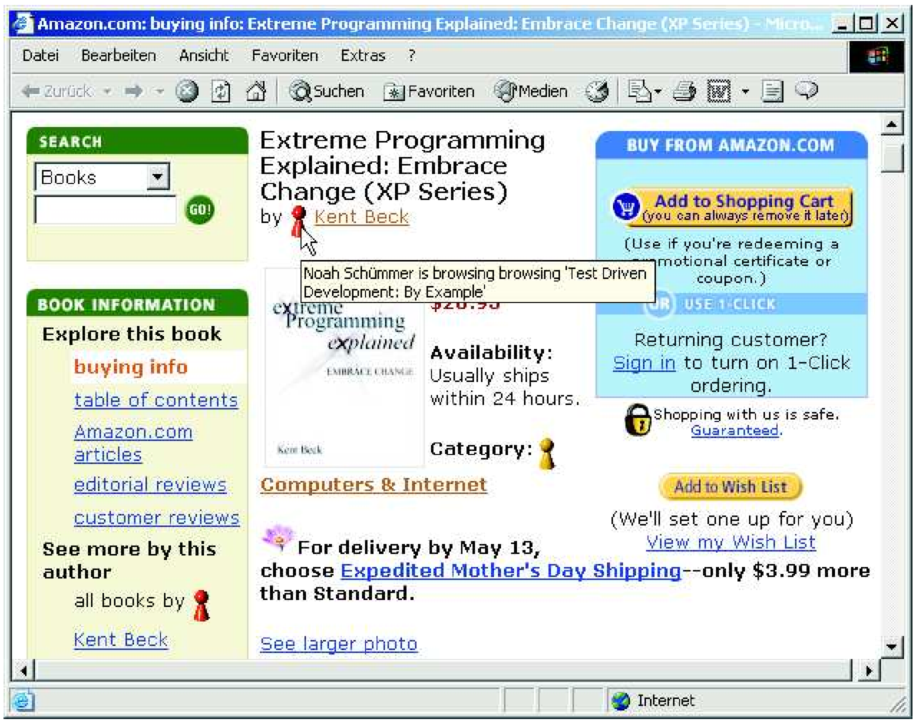
It is up to the application designer to map the different group aspects to the indicator. In general, size, color, and shape can be used as dimensions to represent the above aspects.

A *mapping function* can – in most cases – automate the projection of group aspects to the different dimensions of the indicator.

Rationale

The indicator has two main advantages that help the user to notice the awareness information without being disturbed by it: Firstly, it does not use much space (resp. bandwidth of the information channel) and therefore it can in most cases be integrated with the representation of the artifact. Second, the indicator is visual different to the rest of the displayed information. By carefully keeping the indicator distinguishable from the artifact's visualization, users will no longer mix the awareness information with the shared artifacts.

Figure 10 shows how the awareness information can be displayed using presence indicators. In this design, only one aspect of the confocal user group is reflected in the icon (next to the cursor in figure 10): the color represents the distance of the user (or the user's) to the artifact. In the above case, red (dark) icons represent users who browse very related books whereas green (light) icons represent users who browse more different books.


Amazon.com: buying info: Extreme Programming Explained: Embrace Change (XP Series) - Micro...

Datei Bearbeiten Ansicht Favoriten Extras ?

Zurück Suchen Favoriten Medien

SEARCH

Books

GO!

BOOK INFORMATION

Explore this book

buying info

table of contents

Amazon.com articles

editorial reviews

customer reviews

See more by this author

all books by

Kent Beck

Extreme Programming Explained: Embrace Change (XP Series)

by Kent Beck

Noah Schümmer is browsing browsing 'Test Driven Development: By Example'

Availability: Usually ships within 24 hours.

Category:

Computers & Internet

For delivery by May 13, choose Expedited Mother's Day Shipping--only $3.99 more than Standard.

See larger photo

BUY FROM AMAZON.COM

Add to Shopping Cart (you can always remove it later)

(Use if you're redeeming a promotional certificate or coupon.)

USE 1-CLICK

Returning customer? Sign in to turn on 1-Click ordering.

Shopping with us is safe. Guaranteed.

Add to Wish List

(We'll set one up for you)

View my Wish List

Internet

Figure 10: Graphical in-place visualization of confocal users in a virtual bookstore.

By placing the icon next to the artifact, one directly connects users to artifacts. Especially in settings where the users do not collaborate in the same workspace, this is the only reference point for collaborative activities. This gets even more important, if one addresses the asynchronous case (as stated by Dourish (1997)).

Danger Spots

Ensure that the icon's visualization does not conflict with other visual elements on the screen. Conflicting means in this context that the application itself uses icons of a comparable shape or color. Use a DISTINCT AWARENESS INFO$_{\rightarrow 3}$ in this case.

Known Uses

TUKAN and GAMA-Mall both use little figures to represent confocal or nearby users. GAMA-Mall (Schümmer 2002) was already shown and discussed in the motivation and rationale sections of this pattern.

The programming environment TUKAN (Schümmer 2001b) uses colored figures that are attached to methods to show that other users are working on the same or related artifact (fig. 11).

PoliAwaC (Sohlenkamp, Prinz, and Fuchs 2000) is an awareness client for the PoliTeam collaborative workspace application that was used by members of the German government ministries in Berlin and Bonn. Members exchanged and organized documents using PoliTeam, which were formerly circulated in

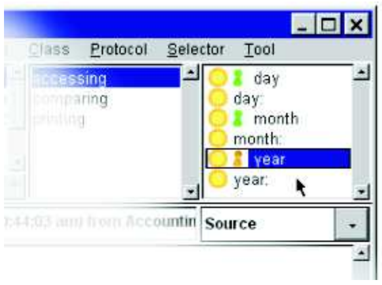
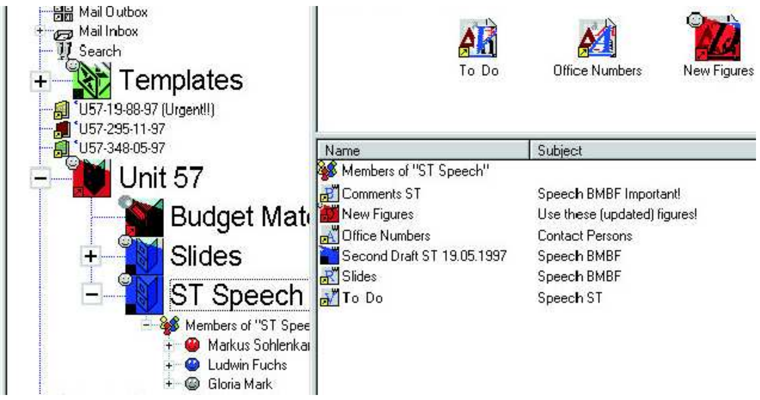


Figure 11: Presence indicators in the TUKAN team programming environment.

folders (when the whole ministry was in Bonn).



Name	Subject
Members of "ST Speech"	
Comments ST	Speech BMBF Important!
New Figures	Use these (updated) figures!
Office Numbers	Contact Persons
Second Draft ST 19.05.1997	Speech BMBF
Slides	Speech BMBF
To Do	Speech ST

Figure 12: PRESENCE INDICATORS in the PoliAwaC awareness-client (Sohlenkamp, Prinz, and Fuchs 2000).

In PoliAwaC, the users can browse shared artifacts using an explorer-like view (fig. 12). If an artifact is currently viewed by another user, the artifact's icon is changed: the size is enlarged to make these documents more prominent and parts of the icon are blended with a color that represents the user who is currently viewing the artifact. A smily icon is finally added to the enlarged icon to visualize the process of viewing.

The system was evaluated in the ministry (on daily use) and the awareness icons were regarded as useful. The only problem that users reported was the mapping between icon-color and user color.

Flying flags as the one shown in the introductionary picture are real-world representations of presence indicators. Whenever the sovereign arrives at the castle, they fly the flag. Thus, anyone can see, whether or not it makes sense to go to the castle and contact the sovereign.

Related Patterns CHANGE WARNING$_{\rightarrow 2.4}$**:** When users manipulate the shared data, it might be more important to report on other user's modifications to the shared data than reporting on other user's presence. The Change Warning pattern does so, by indicating confocal modifications of other users.

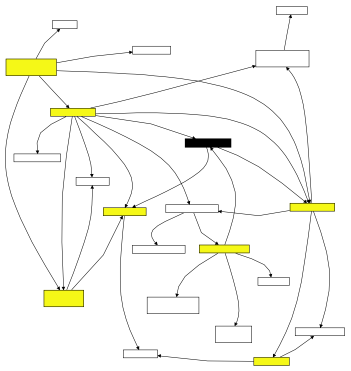
Accumulated Awareness$_{\rightarrow 3}$: When there are many confocal users, it might consume too much screen space to provide presence indicators for all of them. Use the Accumulated Awareness pattern to represent many confocal users with one icon.

Color Scale$_{\rightarrow 3}$ can be used to visualize the fact that the presence indicator displays the presence of a user at a related artifact and not at the artifact to which the indicator is attached. Use this pattern, if active neighbors should be displayed.

Interactive User Info$_{\rightarrow 3}$ describes how to design the awareness info so that it can be used as a means for establishing communication or collaboration sessions.

Distinct Awareness Info$_{\rightarrow 3}$ discusses how the visualization can be designed so that it does not disturb too much from the user's current task.

2.4 Change Warning



Intent

Indicate that the version used by the local user has been changed by another user.

Context

Users work on independent copies of the shared artifacts. The copies are made from the shared artifact, when the user decides to begin working with the artifact. Examples for such a work model are local copies of artifacts drawn from a version management system that uses the check-out/check-in model (like CVS (Price 2000)). Users can select the version that they are interested in and request this version from the version management system. They then work on the local copy. When they have finished their work, they commit the modified version back into the repository.

Most applications working on documents work in a similar way. When the user decides to work with the document, the system loads the document from a storage medium (possibly with shared access) into the application's memory. The users work on the document (in the application's memory) and when they have reached a consistent state, they save the document again to the storage medium.

The system does not lock the shared artifact.

——— ◊◊◊ ———

Problem

While a user works on an independent copy of the artifact, the checkout frequency may be low. So he 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

Imagine two web designers named Fred and George, who author a web site. Since they cannot modify the pages directly on the server, both download all web pages to their local machine on Friday night to do some work on the pages at the weekend.

Fred found out that the user's home pages on the site did not include e-mail addresses. Thus, he decides to create an e-mail list and adds it to the web site. The list contains references to all user's home pages, the user's names, and the e-mail addresses. Additionally, he adds links to the e-mail list to each user's home page. He stores this page on the server at Saturday night.

On Sunday afternoon, George also noticed that the e-mail addresses were missing in his local copy. But he takes another approach to solve this problem: he edits all home pages and adds e-mail addresses to the pages. On Sunday evening, he wants to update the home pages and notices that Fred has also changed the pages. He is angry that he has not worked with the most recent versions and throws away all his changes.

Symptoms *The problem becomes obvious 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.

Solution **Therefore: Provide change warnings in context. This means that the system indicates 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. If changes can be complex, consider to provide a comparison view for the local user's state of the artifact with the remote user's state of the artifact.

◊◊◊

Participants **Local Workspace:** The local workspace contains the local artifacts, which are independent copies of the shared artifacts.

Change Activities are activities (as defined in FROM SHARED DATA TO SHARED WORK$_{\rightarrow 2.1}$) that change artifacts. A change activity contains information on the shared artifact before it was changed, the type of the modification, and the new version of the artifact after the change.

Conflicting Activity: Another user's activity, which modifies an artifact for that the following two conditions hold:

- the activity modifies an artifact, which is also in the local user's local workspace, or
- there are two different sequences $s_1 = \{a_1, \ldots, a_n\}$ and $s_2 = \{a'_1, \ldots, a'_m\}$ of change activities, that both originate from a common artifact version. The local user has an artifact state generated by a'_m while a remote user has an artifact state generated by a_n.

The first possibility is depicted in figure 13. Two users, U_1 and U_2 have both taken an initial version V_1 of an artifact to work on. With activity A_1, U_2 changed the artifact and created a new version V_2 (which is now his new independent

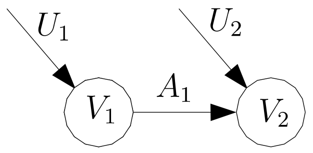


Figure 13: A single activity changing an independent version.

version in his local workspace). User U_2 will see this activity as an conflicting activity.

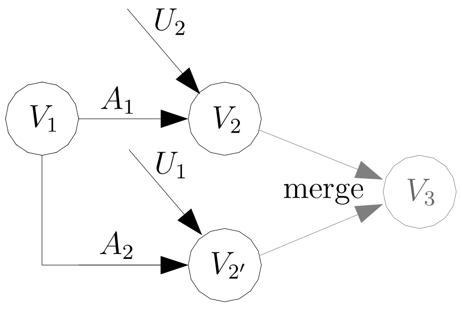


Figure 14: Two activities changing an independent version.

In figure 14, both users have performed an activity on the same artifact. User U_1 has performed activity A_2, which led to $V_{2'}$ and user U_2 has performed A_1 as in the previous example. The two version $V_{2'}$ and V_2 are conflicting and for that reason, the two activities are mutually conflicting activities: A_1 is a conflicting activity for U_2 and A_2 is a conflicting activity for U_1.

Change Warning: The change warning displays the existence of conflicting activities. It can be modelled as an indicator (such as the PRESENCE INDICATOR$_{\rightarrow 2.3}$) or in more obtrusive ways (such as a dialog window).

Rationale

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

From a technical point of view, indicating change warnings changes 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. This is the situation of figure 13. 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 (to V_3 in fig. 14). He can also get in contact with the person, who applied the first change (TALK FIRST$_{\rightarrow 3}$). Both users can then discuss and align their changes and – if considered useful – WORK TOGETHER$_{\rightarrow 3}$ 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. He will have to rethink his actions based on the artifact's changed version.

Danger Spots

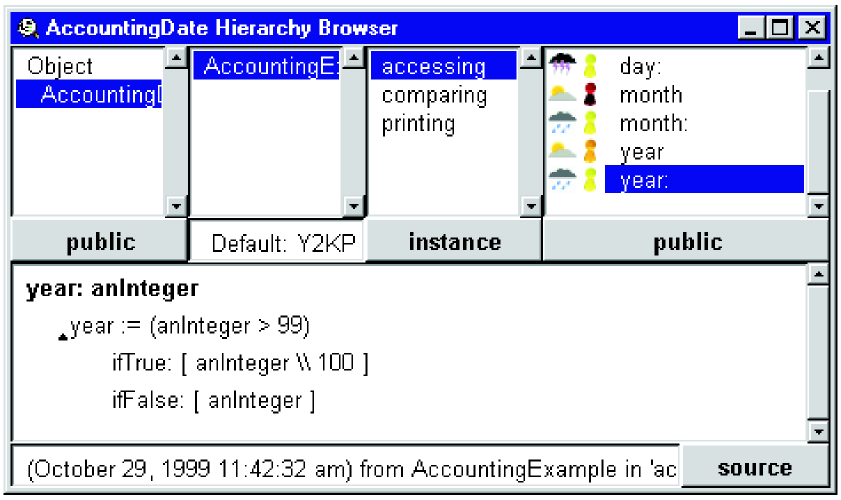
The calculation of conflicting activities may be complicated if users are allowed to select their desired version (which is true for most environments). Consider the case where a user Alice performed a change on an artifact. Later on, she notices that her change was not right. She thus loads the older version of the artifact. But the ELEPHANT'S BRAIN$_{\rightarrow 2.7}$ 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.

Known Uses

TUKAN: The programming environment TUKAN 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$_{\rightarrow 2.5}$ 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 15 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



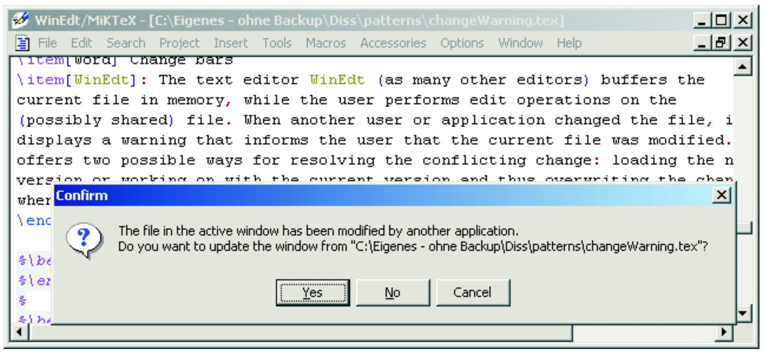
```
year: anInteger
    year := (anInteger > 99)
        ifTrue: [ anInteger \\ 100 ]
        ifFalse: [ anInteger ]
```

(October 29, 1999 11:42:32 am) from AccountingExample in 'ac

Figure 15: Change-warnings in the collaborative software development environment TUKAN.

user may decide to integrate this version before she changes the artifact itself and thus avoid parallel versions.

WinEdt: The text editor WinEdt (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. 16).



```
\item[Word] Change bars
\item[WinEdt]: The text editor WinEdt (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, i
displays a warning that informs the user that the current file was modified.
offers two possible ways for resolving the conflicting change: loading the n
version or working on with the current version and thus overwriting the chan
wher
\enc
```

The file in the active window has been modified by another application.
Do you want to update the window from "C:\Eigenes - ohne Backup\Diss\patterns\changeWarning.tex"?

Figure 16: 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.

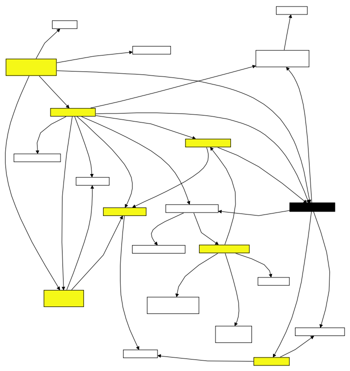
BSCW (Bentley, Appelt, Busbach, Hinrichs, Kerr, Sikkel, Trevor, and Woetzel 1997) displays change indicators for documents that were modified since the last visit. Although it does not model an explicit local workspace, it remembers, which artifacts have been read by a specific user. These versions of 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 ELEPHANT'S BRAIN$_{\rightarrow 2.7}$: Use an ELEPHANT'S BRAIN to store the activities that are used for calculating conflicting activities.

PRESENCE INDICATOR$_{\rightarrow 2.3}$: The presence indicator is comparable to the CHANGE WARNING 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 WARNINGS inform on activities that are completed.

ACTIVE NEIGHBORS$_{\rightarrow 2.5}$ 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).

2.5 Active Neighbors



AKA

Embodiment proximity (Gutwin and Greenberg 2002)

Intent

Visualize activities not only on the local user's current artifact, but also on related artifacts.

Context

You are using local awareness to inform the users on each other's activities. Two or more users are performing work on related artifacts.

The probability of meeting another user on the same artifact is very low since there are much more artifacts than users.

◇◇◇

Problem

The Local Awareness$_{\to 2.2}$ 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.

Scenario

Imagine a virtual bookstore where users can browse and review books. Users can collaborate in the bookstore by chatting on books or browsing for books together. The designer of the bookstore wanted to create a shopping community of interest by bringing together people who like the same books. He decided to use Local Awareness$_{\to 2.2}$ to bring together such users.

He used the Odigo tool (described as a known use in Local Awareness$_{\to 2.2}$) that shows who else is browsing the same web page – in this case the same book.

Some time later, he notices that there are almost never confocal users at most books. The reason for this is that the number of books is very large compared to the number of users who are online at the same time. As a solution, the designer decided to base the selection of confocal users on the whole store (the server part of the book's URL). He used the Odigo mode, which displays all users who are at the store at the same time. But now, the system showed confocal users who had not much in common. For instance, a user looking for computer books was brought in contact with a user looking for crime stories.

Both approaches do not lead to meaningful collaboration.

Symptoms *The problem becomes obvious when ...*

- users still think that it could make sense to work together with other users who share a common *semantic* focus.
- users rarely work on the same artifact.
- the number of artifacts is much larger than the number of users.
- there are semantic dependencies between the artifacts.

Solution **Therefore: Provide awareness on peripheral activities that take place on related artifacts.** Use a SEMANTIC DISTANCE$_{\rightarrow 3}$ 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.

——— ◇◇◇ ———

Participants

Focal Artifact: The artifact that the local user is currently looking for.

Near Artifacts: A set of tuples (a_i, d_i), where d_i is the SEMANTIC DISTANCE$_{\rightarrow 3}$ between the focal artifact and the artifact a_i. The set of focal artifacts includes only those artifacts for that the distance is below the awareness horizon.

Awareness Horizon: The awareness horizon limits the SEMANTIC DISTANCE$_{\rightarrow 3}$ of related artifacts that are still considered as near. The larger the awareness horizon is, the more artifacts are in the set of near artifacts, which implies that probably more users will be in the set of peripheral users.

Peripheral Activities: A set of tuples (a_j, act_j) where a_j appears as artifact in the set of near artifacts and act_j is an activity that takes (or took) place with the artifact.

Peripheral Users: The users associated with the subset of peripheral activities that are still active in case of presence awareness or that trigger a change warning. The set of peripheral users contains those users who are currently working on a near artifact.

Rationale To understand, why peripheral awareness adds new views on group awareness, it makes sense to look at the history of this model: The awareness model used in this pattern is based on the well-known spatial model, which is used to model interaction in virtual environments (Benford and Fahlén 1993). Within the spatial model, artifacts are arranged in a three-dimensional space. Users may interact with artifacts by navigating through the space. Other users can always see where their colleagues are positioned within the space. This is called their presence position (note that a presence position was already used in the LOCAL AWARENESS$_{\rightarrow 2.2}$ pattern). In the spatial awareness model, the local awareness is extended by relating it to a spatial layout of the artifacts: Awareness spreads in space.

Rodden (1996) extended the spatial model by introducing the concepts of focus and nimbus. His focus-nimbus model consists of objects, such as artifacts, and users who are distributed in space. Each object has a well-defined distance to all other objects. Around objects, there is a focus and a nimbus, which are parts of space. The focus includes all objects that are of interest to the object, while the nimbus includes all positions in space where the object might influence other objects. As an example, the focus of a person consists of all people he can see and the nimbus consists of all people who stand next to him. In most cases, interest and influence of an object will fade the further a position in space is away from the object's position.

With focus, nimbus, and presence position as parameters, a suitable awareness function may be defined, whose exact formulation is application dependent (additional examples were provided by Rodden (1996)).

While these two models were mainly developed for virtual (three-dimensional) environments, the active neighbor pattern generalizes the context to any environment, where one can define a semantic distance between two artifacts.

By making the user aware of peripheral users who work on related artifacts, one can ensure that parallel work on semantically related artifacts is detected and coordinated (by a social protocol). A conflict on the other hand is always bound to parallel work on related artifacts that heads in different directions. Since these parallel activities are minimized with the pattern, conflicts are less likely to appear.

In the same way, peripheral users are brought together, which results in collaboration on related artifacts.

Danger Spots

It is important that the awareness horizon is not too large and not too small. A too large awareness horizon will result in two problems:

1. it will contain too many users who worked on artifacts that are not perceived as related artifacts by the user, and
2. it will be hard to calculate, which results in long calculation

times.

A too small awareness horizon will not consider all relevant artifacts.

The following example illustrates, how a reasonable awareness horizon can be found. It is the example of the virtual bookstore, for which average characteristics of the artifacts and the artifacts' use are known.

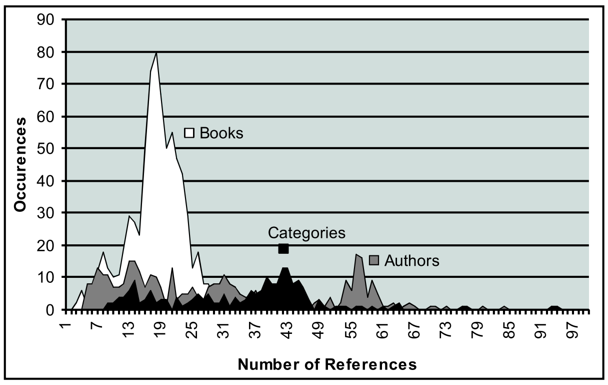


Figure 17: Number of links between artifacts at the Amazon.com bookstore.

Firstly, the average connectivity of the store's artifacts can be calculated by parsing a statistically relevant number of artifacts and searching relations to other artifacts. Schümmer (2002) made these calculations on the basis of 1900 artifacts. Figure 17 shows the average connectivity that was calculated for books, authors, and categories. One can, for instance, see that books link in average to 18.28 other artifacts. The combined average $\bar{l}$ for all kinds of artifacts is 27.5 links. These numbers define the density of the semantic net and are one important factor for calculating the size of the awareness horizon.

The other two important factors are the size of the semantic net (the number of artifacts $\sharp a$ in the shared space) and the number of users $\sharp u$ who are online at the same time. If all these factors are known, one can calculate an awareness horizon dependent of a desired probability that two users meet.

For the virtual bookstore, there are approx. 28 Mio. items[2] in the store. At each second, there are about 12.000 users who visit the store.[3] The system designer can now chose a probability p that users should see other peripheral users in the store. Assume that a designer wants a probability to meet another user of 50%.

In this case, the designer has to ensure that surrounding artifacts overlap with a probability of 50%. In other words: 50% of the artifact space should be in the combined set of all users' near

[2]according to press releases from Amazon.com (2001)
[3]calculated from Internet.com (2001) using the numbers from March 2001 to February 2002

artifacts. The analysis of the bookstore took the simplification that all artifacts were accessed with the same probability. This led to a desired size of the set of each user's near artifacts NA_u of

$$NA_u = \frac{\sharp a \cdot p}{\sharp u} = \frac{28000000 \cdot 0.5}{12000} = 1167$$

From this number, one can calculate an awareness horizon h, so that each activity leads to awareness indicators at NA_u artifacts:[4]

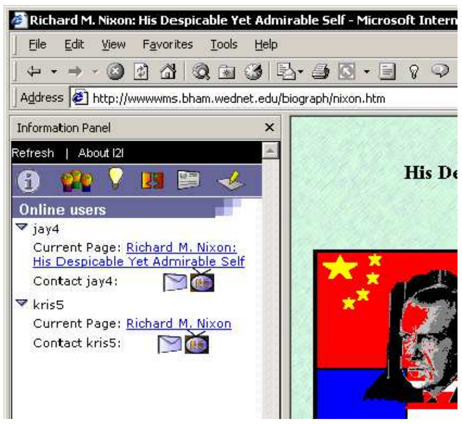
$$\bar{l}^h > NA_u \Rightarrow h > \frac{\ln NA_u}{\ln \bar{l}} = \frac{\ln 1167}{\ln 27.5} = 2.13$$

This number gives a first rough estimation, of the number of indirections that one has to consider. If users should have a chance of seeing other users, one would start with looking at 3 indirections in the semantic net.

Although, the example was shaped to a semantic net for web pages, the formulas are applicable to all kinds of semantic nets (assuming that users use all artifacts with the same probability).

Known Uses

I2I (Budzik, Bradshaw, Fu, and Hammond 2002) is a system that tracks users who access web pages. Besides the current page, it recommends other users for collaboration, who currently read related pages.



http://wwwwms.bham.wednet.edu/biograph/nixon.htm

Online users

jay4
Current Page: Richard M. Nixon: His Despicable Yet Admirable Self
Contact jay4:

kris5
Current Page: Richard M. Nixon
Contact kris5:

Figure 18: Recommendations for collaboration in the I2I system (from Budzik, Bradshaw, Fu, and Hammond (2002)).

CoBrow (Wolf and Froitzheim 1998) displays the presence of peripheral users of web pages as presence indicators. The obtrusiveness of the indicators is dependent of the semantic distance between two users in the document space.

TUKAN (Schümmer 2001b) provides awareness on peripheral users and on peripheral activities that are in conflict with the local user's view of the artifact.

[4]For simplicity reasons, I assume that the distance from an artifact to another directly connected artifact is always 1. The pattern SEMANTIC DISTANCE$\rightarrow$3 discusses this issue in depth.

GroupDesk (Fuchs, Pankoke-Babatz, and Prinz 1995) informs the user on activities that take place on semantically related artifacts. It is a generic groupware architecture that was, for instance, applied in workflow systems.

Related Patterns

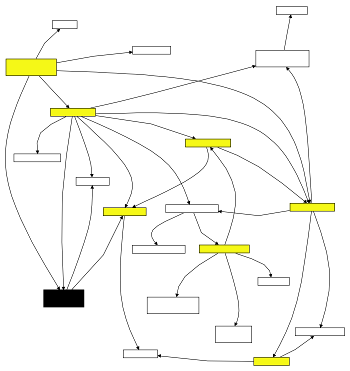
SEMANTIC DISTANCE$_{\rightarrow 3}$: To apply the active neighbors pattern, one has to calculate the distance between artifacts. The semantic distance pattern provides one way of representing distances between artifacts.

SEMANTIC NET$_{\rightarrow 2.8}$: A semantic net can be used to generate a graph structure, by which peripheral artifacts can be calculated.

PRESENCE INDICATOR$_{\rightarrow 2.3}$: The presence indicator can be used to visualize peripheral users. When applying this pattern, ensure that the distance from the focal artifact is part of the visualization. Look at the COLOR SCALE$_{\rightarrow 3}$ pattern to see one way for encoding distances in presence indicators.

CHANGE WARNING$_{\rightarrow 2.4}$ can be used to visualize information on peripheral activities that have modified the artifact.

2.6 Gaze Over the Shoulder



AKA Event Sensors (Prinz 1999)

Intent Detect the users' interaction with shared artifacts.

Context Users interact on shared artifacts using a proprietary tool and want to log activities for future reference. These tools modify shared external data. The results of the modifications are visible within the tool and in the changed shared data.

——— ◊◊◊ ———

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 automatically track user's activities, which you would need to provide awareness.**

Scenario Imagine a seminar, where Alice, Jane, and Sally, a group of students, is preparing a talk. They decided to do literature research independently using the web. Each of them signed up for a section of the topic and starts her search. After two weeks, they meet again and merge the found results. Each of the students has written down some URLs, which she had rated relevant. During the discussion, Jane noticed that one of the topics found by Alice is very related to some information that she encountered during her research but did not rate relevant enough to add it to her URL list.

Unfortunately, she does no longer remember the URL of that page. Since remembering a page required that Jane actively set a bookmark in her browser, she only did this for those pages, that she expected to be relevant. This information was her final result of her single user task – namely the research in her section of the topic.

If she had also remembered the paths that led her to the solution, her team mates could have benefited from this information. On the other hand, if she had manually saved each page, her browsing activities would soon have been very complicated.

Solution **Therefore: Add an additional layer in the communication between the application and the shared data to monitor**

user actions. Allow other parts of your application (e.g. a ELEPHANT'S BRAIN$_{\rightarrow 2.7}$ **or a** LOCAL AWARENESS$_{\rightarrow 2.2}$**) to subscribe to monitored activities.**

◇◇◇

Participants

Request: The *user* performs activities on *shared data*. Activities are internally represented as *requests* from a user interface element (in the broadest sense) to a shared object. An example for a *request* could be the request to save a file in a text editor.

Listener: The listener observes the communication channel and the shared object and forwards information on activities to a subscriber such as the ELEPHANT'S BRAIN$_{\rightarrow 2.7}$. The listener seems to be transparent. That means that the user (resp. the source of the request) does not notice that the request passed a listener when it was transmitted via the communication channel.

Communication Channel: Requests are transmitted using a communication channel. To apply the pattern, one needs to be able to monitor this communication channel.

Since the data is accessible by more than one user, there has to be a communication channel by which the users access the shared data or by which they are informed about state changes.

All communication that takes place via this channel can be monitored (if it is not encrypted) by the listener by either observing state change that results from communication activity or by directly grabbing the information from the communication channel.

In a point-to-point communication, this requires that the listener acts as a *man in the middle*. It receives all requests from the application and forwards these requests to the shared object (thereby playing the role of the original sender of the request).

In a broadcast communication infrastructure, the listener can just act like any other receiver (and listen for the same requests as the intended receiver).

Where the communication channel is located depends on the application's external interaction.

- In WEB applications, it can be the TCP-connection, which is used to transfer HTTP requests. TCP connections can by definition have many intermediaries such as proxy hosts, routers, or gateways. To attach to this communication channel, a listener could act as a proxy. It would perceive a user requesting an URL. On the other hand, WEB applications model most interaction within URLs. The listener would in this case need to parse the

URL (or even the response as well) to retrieve more information on the user's activity.

– In applications that work with files on a shared file system, the communication channel is the file access API, which is often again a network protocol (e.g. in NFS).
It can also be a state change in the file system that indicates that an activity takes place. If users, for instance, start working with a MS-Word document called `MyText.doc`, MS-Word creates a temporary file named `~$Text.doc`, which contains, among others, information on the user who opened the document. When the user closes the document again, this file is removed.
A listener would in this case monitor the directory, where the shared files reside and track the creation and the removal of temporary files. It would perceive
    ▷ a user starting to work with a document,
    ▷ the creation of a new version of the document, and
    ▷ a user ending his work with the document.

Rationale

Since the proprietary tool does not allow other applications to engage in the internal application logic, the only way to distribute state information is by observing external communication channels.

If the observer of the channel notices communication on the communication channel, it can redistribute this information to subscribed users (using, for instance, a shared repository that holds information on all activities, such as the ELEPHANT'S BRAIN$_{\rightarrow 2.7}$). In this way, it is possible, to calculate awareness information without changing the original application.

Danger Spots

Ensure that the user knows that he is monitored. Users have to be able to decide on their own, whether or not they want to be monitored (cf. MASQUERADE$_{\rightarrow 3}$). And they have to feel that being monitored adds value to their activities (e.g. because of the principle of RECIPROCITY$_{\rightarrow 3}$). Otherwise, they will in most cases find a way to do their job bypassing the monitoring mechanisms.

Known Uses

NESSIE: The NESSIE framework (Prinz 1999) provides event sensors, which are comparable to the listener in this pattern. The sensors are placed next to specific artifacts. Whenever an activity is performed on the artifact, the event sensors create an activity object, which is then stored in the NESSIE awareness server (see the description in ELEPHANT'S BRAIN$_{\rightarrow 2.7}$).

User-Centered Push (UCP): Underwood, Maglio, and Barrett (1998) proposed an architecture that monitors the user's interaction with the WWW and makes this information available to other users. Listeners (called *information sleuths*) monitor TCP streams by which a user accesses content on the web.

Whenever content is accessed, information on this access (referred as *fact*) is sent to a shared access log (called *board*).

Proxy-Logger: The Proxy-Logger (Hong and Landay 2001) is a web proxy, which tracks the user's browsing activities to create diagrams that visualize the browsing actions involved in a specific task. The listener is modeled as a proxy. Users enter the URL, which they want to visit, in the proxy's welcome page. The proxy then modifies all delivered HTML content, so that all subsequent URL's are also retreived by the proxy.

GAMA Mall (Schümmer 2001a) also uses a proxy approach to monitor web browsing activities.

Related Patterns ELEPHANT'S BRAIN$_{\rightarrow 2.7}$ provides means for storing monitored activities.

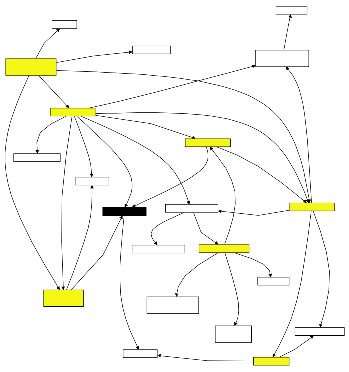
MASQUERADE$_{\rightarrow 3}$ provides the user with means to switch the monitoring mechanisms on and off.

RECIPROCITY$_{\rightarrow 3}$ is an important issue when users are asked to provide personal information. Make sure that the user perceives a personal benefit by providing personal information.

PROXY: The most obvious method for implementing the GAZE OVER THE SHOULDER pattern is to use a Proxy pattern (Rohnert 1995) in a way that the communication channel is redirected through the proxy. But the proxy approach is only one possibility to keep track of actions. As discussed in the participants section, there are different points, where the listener can be attached to the communication channel. Which point is the most appropriate depends on the nature of the communication channel.

NAVIGATION OBSERVER (Rossi, Garrido, and Carvalho 1995) solves a comparable problem in the context of hypermedia applications. As in GAZE OVER THE SHOULDER, actions are recorded for future reference. The main difference is that the NAVIGATION OBSERVER implements a full OBSERVER pattern (Gamma, Helm, Johnson, and Vlissides 1995) with the implication that update messages are sent from within the application directly to the observer. The application (i.e. the controller) thus needs to know its observers that are notified on state changes. This is in conflict with one constraint of the GAZE OVER THE SHOULDER'S problem: *the application should not be changed.* The observation mechanism in GAZE OVER THE SHOULDER is in contrast independent of notification mechanisms of the model.

2.7 Elephant's Brain



AKA Event History, Activity Log

Intent Store information on the users' activities in a log to allow the users to understand (other) users' activities and the artifacts' evolution.

Context Users perform parallel activities on shared artifacts without being totally sure about the effects. They tend to try out solutions without knowing, whether or not the effects will be correct. Another issue is that they are not always able to tell which artifact they will use for their activity, and often activities conflict with other users' activities.

Users work with the shared artifacts for a long time and it may be necessary to review artifacts that were manipulated in a former work episode (e. g. the programming work of the previous week, month, etc.).

◇◇◇

Problem **Merging two user's (past or current) work is a difficult task. It requires 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.**

Scenario Imagine a single user software development environment. The programmer is developing an application using a development approach, where testing and development are two alternating phases. She begins with writing a test for the desired functionality – assume that she is looking for an algorithm that calculates all prime numbers. In the test, she checks, whether the algorithm finds the prime numbers 3, 5, and 7.

She then continues to write software that fulfills the test - let's say that the algorithm just checks, whether the number is larger than 1 and odd.

In a next step, she creates a second test, that ensures that 9 is not a prime number. Writing code to pass the test brings her to the idea of changing the code in a way that it checks, whether the number is odd and between 3 and 7.

A next test demands to also accept 11 as a prime number. At this point, the programmer notices that the last change was not really correct. Thus, she has to revert the change. But unfortunately the development environment does not support her. She has to remember the change that she applied and try to recover the previous code version from her memory. It may be simple if the action that needs to be reverted is the last action that she performed. But often, one does not see the problems of a specific activity directly after the activity was performed. It is rather often the case that one identifies the effects of wrong activities a long time after the activity was performed.

Now consider a scenario, where two programmers work on the same project. Each programmer solves different problems, but both problems involve the same shared artifact. Even if the first programmer does still remember, which changes she performed, the second programmer cannot revert these activities, because he does not know, what his colleague did.

Symptoms *The problem becomes obvious when ...*

- users often notice that an artifact changed, but could no longer remember how it looked before.
- users cannot detect who performed incorrect changes, and what these changes were.
- users cannot find out, what another user X did to the artifact Y yesterday.
- users don't understand the changes of other users.

Solution **Therefore: 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.

◇◇◇

Participants Two main participants are involved in the pattern: the **log** and the **activity**. A log is a container that holds the activities. Logs should be persistent to allow the users to refer to activities of previous sessions.

The **activity** includes information on

- the *type* of the activity, such as reading, editing, creating, removing, etc.,

- the *artifact* that was touched by the activity (often in two versions: the version before the activity and the version after the activity),
- the *time* at wich the activity took place (often as a set of two timestamps representing the start and the end of the activity), and
- the *user* who performed the activity.

In several systems, activities further include a user comment, which provides more information on the activities intent. Activities should comprise one semantic user transaction with the system. Examples could be the selection of a menu command, the insertion of a new paragraph in a text document, or the viewing of a web page in a web browser. Activities should be updated immediately, when a user interacts with the application.

Clients interact with the log by adding and consuming or querying activities. The log should be accessible from all clients who need to perform calculations on it or who submit activities for log storage. In collaborative applications, this implies that it can be accessed using a *network protocol*.

For the display of up-to-date collaboration information, the log can serve as a Publisher of new activities according to the PUBLISHER-SUBSCRIBER pattern (Buschmann, Meunier, Rohnert, Sommerlad, and Stal 1996). Clients who use the log to provide awareness information, subscribe to activities regarding specific artifacts or initiated by a specific user. They will be informed, whenever the set of activities changes that matches the subscription pattern.

Rationale

By logging all activities that users perform, one can inspect all activities later. This helps to understand the evolution of a specific artifact (by looking at all activities that took place on the artifact) or a specific user's work (by looking up all activities, which were initiated by a specific user).

Since the log is persistent, the information will be remembered also when the user has forgotten it.

The ELEPHANT'S BRAIN goes beyond version management functionality in two main points:

1. It also remembers accesses that did not change the artifact. These accesses are not mandatory for restoring the system at a specific point of time. But they help to understand the user's background for a specific activity, and in some cases, they help to reveal conflicting activities.
2. The ELEPHANT'S BRAIN provides means for notifying interested users on activities, which take place with a specific ar-

tifact (note that some version management systems support this, which is the reason why CVS is included in the known uses section, although it does not remember non-modifying accesses, as it was discussed in the previous paragraph).

Danger Spots

Ensure that many users can add activities to the log at the same time. One way for achieving this is to model the log as a bag to which users can add activities in any order producing the same result. Since each activity is unique (distinguishable user and time), there will be no conflicts in adding activities to the log.

When referencing artifacts in the activity description, you have to make sure that the right version of the artifact is referenced. In environments, where no versioning is available, you should include sufficient information to restore all different versions of an artifact from inspecting all activities that took place on the artifact.

Prinz (1999) argues that the activity log should be decoupled from the application, which generates events. This allows an easy extension to support a large variety of events generated by different client applications.

Note that the ELEPHANT'S BRAIN can only store those activities that it gets informed of. For off the shelf tools like CAD packages, this might just be the creation of a new version. You should therefore carefully examine the application to monitor as much information as possible since this information is crucial for a detailed (and intelligent) group support. Refer to GAZE OVER THE SHOULDER$_{\rightarrow 2.6}$ for hints on how to gather such information.

Known Uses

CVS version logs: The CVS version management system (Price 2000) maintains a file `history`, where meta-information on all activities is stored. This information includes the user, the touched artifact, and the type of the modification. The version information can be obtained using the `cvs history` command. An example output looks like this:

```
cvs history -c
M 2003-02-27 13:09 schuemm 1.2 README    test == ~test2\test
A 2003-02-27 13:14 schuemm 1.1 short.txt test == ~test2\test
M 2003-02-27 13:16 schuemm 1.2 short.txt test == ~test2\test
```

Each line in the history output describes one user action within the CVS repository. It shows a type-code of the activity, its time, the user who performed the activity, and the artifact that was modified.

One can configure a cvs server to run any software when artifacts are accessed. This can be used to allow distributed clients to subscribe to changes according to ELEPHANT'S BRAIN. A subscription mechanism as it was proposed by the PUBLISHER-SUBSCRIBER pattern is however not part of the standard cvs system.

CVS only logs activities, which access the repository. This is sufficient to support a Change Warning$_{\rightarrow 2.4}$. For the provision of a Presence Indicator$_{\rightarrow 2.3}$, it is too few information. Presence indicators would need information not only on those activities that were performed to copy shared artifacts to a local workspace. It would also need the information on what the users do in their private workspace.

VisualWorks Change Set: The VisualWorks Smalltalk environment (Cincom 2001) writes all changes to a change file. This change file is mainly intended to recover from system crashes, but an important additional use-case is the inspection of changes of an artifact.

VisualWorks provides a special changes browser for this, which shows all changes of a specific source artifact.

Since the programming environment is a single user environment, it does not provide any multi-user access to the change file.

NESSIE Awareness Server: The NESSIE Awareness Server (Prinz 1999) is a general purpose awareness server, which stores events. Each event carries information on the originator, the action, the touched artifact, and the time at that the event took place. The events are implementations of Elephant's Brain's activities.

Client applications can add activities to the server using an HTTP-based interface. They can also subscribe to changes by specifying the type of the activity and a desired context (i.e. the touched artifact's location).

BSCW (Bentley, Horstmann, and Trevor 1997) is a shared workspace system that logs all activities, such as reading, editing, or moving documents, that are performed in the shared workspace. The logfile captures events that look like this:

```
User:[52, 'Alice']
object:[134, 'Home']
Type:CopyEvent
Time:1030181775.14
Path:[[63, '&NewHome']]

User:[51, 'Bob']
object:[124, 'BusinessPlan.doc']
Type:CutEvent
Time:1030182090.44
Path:[[119, 'Workgroup']]
```

The example shows parts of two activities. First, a user *Alice* has copied an artifact called *Home* to a folder *NewHome*. In the second activity, *Bob* cut the artifact *BusinessPlan.doc*.

TUKAN (Schümmer and Schümmer 2001) is a collaborative programming environment that extends VisualWorks Smalltalk.

It logs all activities that users perform in the environment (e.g. reading source code or modifying a class file) and stores these activities in an activity log.

The activity log is stored as a shared object, which is replicated to all clients using the COAST framework for synchronous groupware (Schümmer, Schümmer, and Schuckmann 2001). The replication mechanisms also include a distributed version of PUBLISHER-SUBSCRIBER to trigger view updates or other actions when any log entry changed.

Related Patterns

PROXY OBJECT$_{\rightarrow 3}$: If the artifacts cannot be accessed directly in the application, one has to use a meta representation in the activity description.

GAZE OVER THE SHOULDER$_{\rightarrow 2.6}$: When the application's internal control flow cannot be extended to monitor activities, use the GAZE OVER THE SHOULDER pattern. It will seek for activities, which are then forwarded to the ELEPHANT'S BRAIN's log for future reference.

LOCAL AWARENESS$_{\rightarrow 2.2}$: Use Local Awareness to inform users of different activities that currently take place at the same artifact to avoid conflicting work. After the activity was performed, remember it to support users in resolving previous conflicting activities.

MODEL-VIEW-CONTROLLER: If the application is implemented following the MODEL-VIEW-CONTROLLER pattern (Krasner and Pope (1988) and Buschmann, Meunier, Rohnert, Sommerlad, and Stal (1996)), one way of determining activities is to hook into the control flow of the controller. Whenever the controller receives a startUp message (i.e. when it starts its work), an activity is created. The activity is then filled with artifacts that are accessed while the user uses the controller. When the controller's control-flow terminates, it fills the end time of the activity.

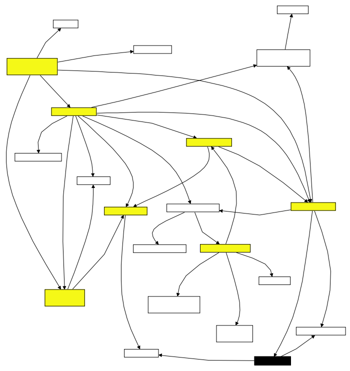
COMMAND: One can interpret the activities as *Commands* according to the COMMAND design pattern (Gamma, Helm, Johnson, and Vlissides 1995). The main difference is that commands should be able to execute (and undo) themselves, which exceeds the simple logging purpose of the activities. Commands are therefore more tightly bound to the application. In contrast to activities, they are potentially active and not just descriptive. If the application uses COMMANDs, one can reference to commands as activities and store them in the log.

CHANGE LOG: A change log (Anderson 2000) stores different states of an object or an object's attribute together with additional information on the originator of the change. It is comparable to ELEPHANT'S BRAIN because it also stores old states of the artifact and thus provides the information, which is needed to reconstruct past activities. But the focus of both patterns is different: A change log is mainly solving the problem of restoring or accessing old state of an object whereas the elephant's brain focusses on logging any activities (not necessarily modifying accesses).

EDITION: The edition pattern (Anderson 2000) shows, how the change of an object's state can be associated with the event that caused the change. It directly binds the activity to the object, which was affected by the activity. For cases, where the shared objects can be manipulated, this is an alternative approach to store the activities. The decision, whether to store the activities (on any shared artifact) in a repository (ELEPHANT'S BRAIN) or directly with the artifact (Edition) depends on the access patterns for the information. If activities are mainly accessed by time or user, they are easier to find in a repository. If they are accessed for a specific artifact, they can also be stored directly with the artifact. For the collaborative setting of ELEPHANT'S BRAIN, one should in any case ensure that not only modifying accesses are stored (as it is the case in the original EDITION pattern).

SHARED REPOSITORY: The log should be modelled as a shared repository (Mularz 1995) to keep it open for future additional applications as well as future additional awareness views.

2.8 Semantic Net



Intent

Use a semantic net, which contains all artifacts and the relations between artifacts to make semantic relations explicit.

Context

You decided to use the Active Neighbors$_{\rightarrow 2.5}$ pattern to provide awareness on activities that take place at related artifacts. To apply the awareness model proposed by Active Neighbors$_{\rightarrow 2.5}$, you need to find out, how distant two artifacts are, especially, which other artifacts are near to a focused artifact.

◇◇◇

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.

Scenario

Imagine Bob and Claire, two authors of a music magazine, who work on essays for the next issue of the magazine. Bob is currently working on an essay on Ludwig van Beethoven's sixth symphony. For that sake, he needs to do much research on the development of this work. Claire is at the same time writing an essay on Franz Schubert's first symphony.

Both Beethoven's and Schubert's works could have been related by many reasons. For instance, both are of the genre symphony and both were written in the classical period of music. But the essays of Bob and Claire are not directly related because they don't share many keywords.

Applying the Semantic Distance$_{\rightarrow 3}$ pattern will therefore result in a long distance between Bob's and Claire's work (since they work on topics that are not directly related). Both users will not find relations to the other user's work in the system, although they might benefit from cooperation because Beethoven and Schubert share a common context.

Symptoms *The problem becomes obvious when …*

- too few relations are provided for ACTIVE NEIGHBORS$_{\rightarrow 2.5}$, which results in the absence of peripheral users.
- the calculation of similarity between two artifacts is too time consuming, which leads to long delays when near artifacts are to be displayed.
- users more often say that their work is semantically related even though the system does not propose this. This may be the case because of the fact that the users combine the two artifacts with their background knowledge or by interfering the artifact's contexts.

Similarity mechanisms produce large distances, if the two artifacts have few in common. This is the case for most artifacts in a large set of artifacts. Regardless the high probability for large distances, one has to calculate the distances between any artifact to find out their distance. This is time consuming, which implies that it cannot be applied in an interactive system, where artifacts change often.

But the desired information is only needed for artifacts that have a short semantic distance. Thus, computation is spent on results that will not be used.

Similarity mechanisms also often fail, if the semantic relation is only provided via an intermediate artifact (if an artifact a is close to an artifact b, which is in turn close to an artifact c, it does not imply that similarity mechanisms would relate a with c).

Solution **Therefore: 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$_{\rightarrow 3}$ **pattern). Define the distance between two artifacts as the length of the shortest path between these artifacts.**

——— ◇◇◇ ———

Participants **Semantic Net:** The semantic net represents semantic relationships between artifacts. The rationale section discusses in depth, how semantic nets work.

It calculates the semantic distance between two artifacts by finding the shortest path between these artifacts. The shortest path between two artifacts can be calculated using any shortest path algorithm from literature (like Dijkstra's algorithm (Dijkstra 1959)). For a very large set of artifacts and a number of connections that is much smaller than the number of possible connections, this is normally faster than calculating the distance between each pair of artifacts using a SEMANTIC DISTANCE$_{\rightarrow 3}$. Especially, the shortest path algorithms can be modified as follows to only calculate paths that are shorter than an upper bound, which is defined by the *Awareness Horizon* of the ACTIVE NEIGHBORS$_{\rightarrow 2.5}$ pattern:

```
$M_1$ : set of not yet visited artifacts and their distance to the source artifact $a_0$
$M_2$ : set of visited artifacts and their distance to the source artifact $a_0$
$h \leftarrow$ awareness horizon
$M_1 \leftarrow M_1 + (a_0, 0)$
while $M_1 \neq \{\}$ do
  pivot $\leftarrow (a, d_1) \in M_1$ so that there is no $(b, d_2) \in M_1$ with $d_1 > d_2$
  $M_1 \leftarrow M_1-$ pivot
  $M_2 \leftarrow M_2+$ pivot
  $d \leftarrow$ distance of pivot
  $a \leftarrow$ artifact of pivot
  for all $n$ where $n$ is a neighbor of $a$ do
    $d_2 \leftarrow d+$ distance of $n$
    if $(d_2 \leq h) \wedge (\neg(n \in M_1) \vee ((n \in M_1) \wedge (d_2 <$ distance of $n$ in $M_2)))$ then
      $M_1 \leftarrow M_1 + (n, d_2)$
    end if
  end for
end while
```

Artifact: The artifact represents a node in the semantic net. Whenever an artifact is added to the semantic net, it is parsed regarding relations to other artifacts.

If the other artifact does not yet exist in the semantic net, one can use a PROXY OBJECT$_{\rightarrow 3}$ to represent this artifact. The PROXY OBJECT should also be used if the original representation of the artifact cannot be modified to store relations to other artifacts.

The artifact has an accessor method, which can be used to obtain all relations to other artifacts.

Relation: A relation is an edge in the semantic net. If two artifacts are semantically related, there exists a relation in the semantic net, which is weighted according to the SEMANTIC DISTANCE$_{\rightarrow 3}$ between the artifacts. If no semantic distances are used, it is set to a constant value representing the type of the relation.

Rationale

The use of semantic nets has two basic advantages:

1. it localizes the calculation of relations (which enhances the overall performance), and
2. it captures semantic relations that exist only because of an intermediate artifact (which produces more exact results).

The first issue enables interactive applications to do this calculation on the fly. Whenever a new artifact is added to the semantic net, one has to extract “important” properties from this artifact and relate the artifact to other artifacts, which represent these properties. The technique for this can often benefit from the inherent semantic of the application.

Consider, for instance, a semantic net representation of software artifacts. If the artifacts are source code fragments of an object oriented system, one can interpret the artifact to directly get a list of related artifacts. In the example, one would parse, for instance, a method to find out,

- to which class the method belongs,
- which other classes are used by this object,
- which attributes are accessed within the object,
- which local variables are defined in the method and of which type they are, or
- which other methods are called from within the method.

This can be done by solely inspecting the artifact. Any indirect relations can then be detected by traversing the semantic net.

The runtime calculation cost of semantic distances is reduced to the problem of finding a shortest path between two artifacts (and using the sum of the relation distances as the total distance between the two artifacts).

Compared to a complete comparison of all artifacts with a runtime behavior of $O(|A|^2)$, the above algorithm has a runtime behavior of $O(|E| * |A'|)$ where A' is the number of artifacts within the awareness horizon and $|E|$ is the number of relations that have an artifact in $|A'|$ as their source. Since A' is in most cases much smaller than $|A|$ and the semantic net has only sparse edges, the runtime behavior is much better than a complete comparison.

The use of a semantic net enables the system to detect relations between two artifacts a_1 and a_2 that are present because of an intermediate artifact a_3, to which each of the two artifacts is directly related. Since the relations $a_1 \rightarrow a_3$ and $a_3 \rightarrow a_2$ have a finite SEMANTIC DISTANCE$_{\rightarrow 3}$, the shortest path between a_1 and a_3 is no more infinite (it is the sum of the two distances).

An early introduction of the concept of semantic nets is given by Quillian (1968) who created the theory that the human mind organizes its memories by concepts, which are modelled as a semantic network. An often cited example is a semantic net that describes the concept *Canary* (cf. figure 19). It is related to the concept *Bird*, which has *Feathers*. *Birds* are *Animals* and *Animals* have *Skin*. Quillian found out that it is easier for the human brain to associate canaries with feathers than with skin. A reason for this

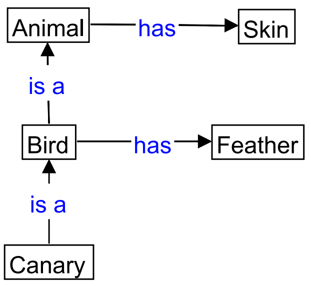


Figure 19: An example for a semantic net.

is that the distance in the semantic net is longer between *Canary* and *Skin*.

Thus, modelling application data as semantic nets will in most cases create semantic structures that are comparable to the human understanding. Exploring such a structure reflects – to some extent – the conceptual knowledge of the artifacts.

Danger Spots

The more indirections are used to calculate the distance between two artifacts, the harder it gets for the user to understand the semantic relation between two artifacts in the semantic net. In such cases, it may help to explain the path that links to artifacts to the user.

Known Uses

TUKAN Within TUKAN (Schümmer 2001b), all Software artifacts are represented as nodes in a graph. Directed relations state semantic dependencies between two nodes. TUKAN calls the semantic net *software space*. Relations are mainly created by parsing the artifacts of the software development activities. The example provided in the rationale section explains one instance of parsing, as it is done in TUKAN.

K-Infinity (Intelligent Views 2003) is an application to model semantic networks. The networks can be associated with the company's artifacts (documents) to reveal relations between artifacts. To generate the semantic net, the user first has to define a set of core concepts (manually) and tell, how these concepts are related. Documents can then be related to the core concepts using a mark-up-tool. Note that this is done locally for the artifact. The author has to look at the artifact (the text document) and define, which text passages reference which concept.

Figure 20 provides an example of a semantic net in K-Infinity. It shows, how nuts (displayed in the right content-window and labelled "Nuß") are related to other plants and artifacts explaining these plants. They are related to a book called "Obst und Gemüse" (on the top left corner of the net), which discusses fruits and vegetables in general.

K-Infinity is one representative for the class of applications

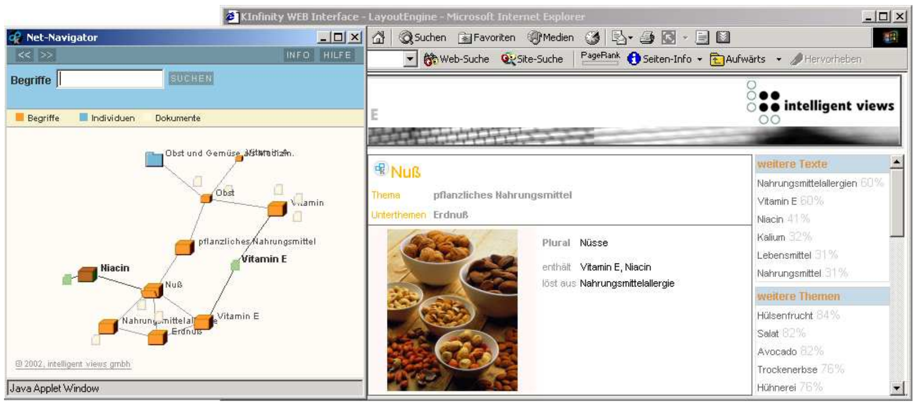
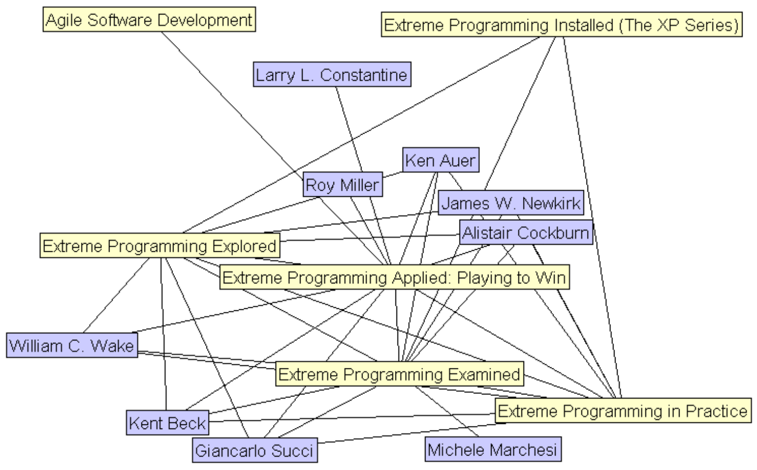


Figure 20: Semantic Nets in K-Infinity (online demo at http://www.i-views.de/web/).

that use semantic nets for knowledge management. Other systems include WordNet®(Miller, Beckwith, Fellbaum, Gross, and Miller 1990), a system that models thesauri by means of semantic nets.

GAMA Mall (Schümmer 2001a) maps artifacts of a virtual bookstore to a semantic net. The web pages of the store are parsed whenever they are requested by the user. The parser searches for references to related books, which appear in a special section of the page, to authors, which always have a specific URL format, or to categories that are also detected by a special URL format.



Agile Software Development
Extreme Programming Installed (The XP Series)
Larry L. Constantine
Ken Auer
Roy Miller
James W. Newkirk
Alistair Cockburn
Extreme Programming Explored
Extreme Programming Applied: Playing to Win
William C. Wake
Extreme Programming Examined
Extreme Programming in Practice
Kent Beck
Giancarlo Succi
Michele Marchesi

Figure 21: Semantic net of the GAMA-Mall bookstore.

Figure 21 shows a small part of the semantic net of the GAMA-Mall bookstore. The nodes represent artifacts. The nodes' colors represent the artifacts' types. Darker nodes represent authors, while lighter nodes represent books. GAMA-Mall did tolerate fuzziness of relations in the semantic net. In figure 21, this is, for instance, the relation between the author

"Kent Beck" and the book "Extreme Programming Examined". Kent Beck has not been an author of this book, but he is referenced on the page in the book shop that shows the book. The reason for this is that Kent Beck wrote a book, which is recommended on the page for "Extreme Programming Examined". For the purpose of providing awareness on peripheral users, this was not problematic since the semantic relation still exists (the recommendation of Amazon also carries some semantics).

GroupDesk (Fuchs, Pankoke-Babatz, and Prinz 1995) uses semantic nets to model relations between artifacts in groupware applications. The authors provide an example of an electronic circulation folder, which relates documents with the station at which they have to be processed. The network is used to propagate awareness notifications (information on activities on the artifacts) among the different nodes in the semantic net.

CoBrow (Wolf and Froitzheim 1998) uses semantic nets to calculate distances between users who use HTML documents in the web.

Related Patterns SEMANTIC DISTANCE$_{\rightarrow 3}$: Calculates the distance between two given artifacts. It can complement the SEMANTIC NET pattern. In this case, SEMANTIC DISTANCE calculates the distance that is attached to the relation in the semantic net.

PROXY OBJECT$_{\rightarrow 3}$: The PROXY OBJECT is used to store relations to other Artifacts, whenever the original artifact can not be modified.

3 Additional Thumbnails

As mentioned in the introduction, this paper only included parts of the GAMA pattern language. All patterns that were not described in a prosaic way will be presented as thumbnails in this section. The patterns are listed in alphabetic order.

Accumulated Awareness: Merge activity indicators of several users in one indicator to save screen space.

Color Scale: Use a color code to distinguish important information (with an obtrusive color) less important information.

Distinct Awareness Info: Ensure that the visualization of the awareness info has a large visual difference to the visualization of the application data. Make it less obtrusive than application data.

External Awareness View: Provide an external awareness information in case that the visualization of the application cannot (or should not) be changed.

Interactive User Info: Make the information about other users clickable and connect it with means for communication and collaboration.

In-Place Awareness View: Show the awareness information in a close proximity to the related artifact to ease the process of connecting awareness information with the semantic context of the referred artifact.

Masquerade: Control, how much private information you reveal to other users when interacting in a collaborative environment.

Proxy Object: Store meta-information regarding the application's artifacts without changing the artifacts.

Radar View: Shows a graphical representation of the nimbus of the local user.

Reciprocity: Ensure that the users benefit, if they contribute to the system. Let the benefit grow, when the user contributes more.

Semantic Distance: Reflect semantic importance in the layout of the semantic net.

Spatial Domain Model: Ease understanding of a semantic net by using the metaphor of a virtual space, in which the net is laid out.

Talk first: Communicate with a related user to negotiate the ways of collaboration.

Time Compressor: Show the presence of other users for longer than they are present.

Work Together: Accomplish an activity together with another user.

4 Conclusions

Computer support for dynamic teams gain an increasing importance while flexible processes and organizational models become widely accepted. Since collaborative work always involves interaction among users in their workplace, I argue that these users should play an important role during the design of CSCW systems that support dynamic teams.

The proposed GAMA pattern language can be an important means for both, designers and end-users, to explore the different design options and learn from best practices in the field. Although this language is still in an evolving state, the application of first patterns showed that they could be successfully applied by non-experienced software developers with no background in the research of collaborative systems.

Current work on the GAMA language is twofold: on one hand, the patterns need to be refined to complete the language. On the other hand, the patterns should be applied to see whether or not non-experienced developers and end-users are able to successfully design collaborative systems with the patterns (even though first experiences showed that also the uncompleted patterns were a valuable means for training developers). In this context, the preliminary versions of the patterns are currently applied in a project, where a JAVA programming environment is made collaboration aware.

4.1 Acknowledgements

I would like to thank all colleagues who discussed parts of the GAMA language and provided valuable hints for improving the language as well as individual patterns. These are Joerg Haake (FernUniversität Hagen – Germany), Alejandro Fernàndez and Torsten Holmer (Fraunhofer IPSI – Germany), and Robert Slagter (Telematica Instituut – The Netherlands). Especially, I'd like to thank my EuroPLoP shepherd Kristian Elof Sørensen for being patient enough to accept yet another pattern for shepherding every other week and finally challenging me to complete the language to the current state. Valuable comments have been provided by the participants of the EuroPLoP2003 writer's workshop. I'd like to thank all participants for their constructive feedback.

Special thanks are due to the co-organizers of the CSCW 2002 workshop on Socio-Technical Pattern Languages (Catalina Danis, Sharon Greene, and John Thomas, IBM – US) for the valuable discussions on pattern structures and specialties of patterns in the area of CSCW. As the language grows, this list will hopefully grow as well.

References

Alexander, C., S. Ishikawa, and M. Silverstein (1968). *A pattern language which generates multi-service centers.* University of California, Berkeley: Center for environmental structure.

Amazon.com (2001). Amazon.com introduces worldwide digital group.

Anderson, F. (2000). A collection of history patterns. In N. Harrison, B. Foote, and H. Rohnert (Eds.), *Pattern Languages of Program Design 4*, pp. 263–297. Reading, MA, USA: Addison-Wesley.

Beck, K. (1999). *eXtreme Programming Explained.* Reading, MA: Addison Wessley.

Benford, S. and L. Fahlén (1993). A spatial model of interaction in large virtual environments. In *Proceedings of the Third European Conference on Computer-Supported Cooperative Work (ECSCW '93)*, Milan, Italy, pp. 109–124. Kluwer.

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.*

Bentley, R., T. Horstmann, and J. Trevor (1997). The world wide web as enabling technology for cscw: The case of bscw.

Bradshaw, S., J. Budzik, X. Fu, and K. J. Hammond (2002). Clustering for opportunistic communication. In *Proceedings of WWW 2002.* ACM Press.

Budzik, J., S. Bradshaw, X. Fu, and K. J. Hammond (2002). Supporting online resource discovery in the context of ongoing tasks with proactive software assistants. *International Journal of Human-Computer Studies 56*(1), 47–74.

Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal (1996). *Pattern-Oriented Software Architecture: A system of patterns.* Chichester, West-Sussex, UK: John Wiley and Sons.

Cincom (2001). *VisualWorks Application Developer's guide.* Cincinnati: Cincom Systems, Inc.

Cohen, D., M. Jacovi, Y. S. Maarek, and V. Soroka (2000). Collection awareness on the web via livemaps. In *CSCW'2000 Workshop on Awareness and the WWW*, http://www2.mic.atr.co.jp/dept2/awareness/.

Coldeway, J. (2003). Interaction patterns of agile development. In A. O'Callaghan, J. Eckstein, and C. Schwanninger (Eds.), *Proceedings of the 7th European Conference on Pattern Languages of Programs, EuroPLoP'02*, Irsee, Germany, pp. 329–342. UVK.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 269–271.

Dourish, P. (1997). Extending awareness beyond synchronous collaboration. In *Position paper for the ACM CHI'97 Workshop on Awareness in Collaborative Systems*, Atlanta, Georgia.

Fuchs, L., U. Pankoke-Babatz, and W. Prinz (1995). Supporting cooperative awareness with local event mechanisms: The groupdesk system. In *Proceedings of ECSCW 1995*, Stockholm, pp. 247–262.

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

Gross, T. (1999). Supporting awareness and cooperation in digital information environments. In *Presented at Basic Research Symposium at the Conference on Human Factors in Computing Systems – CHI'99*, Pittsburgh, PA, pp. online at: http://orgwis.gmd.de/g̃ross/publ/chi1999/gross_chi1999brs.html.

Gutwin, C. and S. Greenberg (2002). A descriptive framework of workspace awareness for real-time groupware. *Comput. Supported Coop. Work 11*(3), 411–446.

Hinds, P. J. and J. Pfeffer (2003). Why organizations don't "know what they know": Cognitive and motivational factors affecting the transfer of expertise. In M. S. Ackermann, V. Pipek, and V. Wulf (Eds.), *Sharing Expertise: Beyond Knowledge Management*, pp. 3–26. Cambridge, MA, USA: MIT Press.

Hong, J. I. and J. A. Landay (2001). Webquilt: A framework for capturing and visualizing the web experience. In *Proceedings of WWW10*, Hong Kong, China, pp. 717–724.

Intelligent Views (2003). Company homepage. http://www.i-views.de.

Internet.com (2001). Cyberatlas: Traffic patterns.

Krasner, G. E. and S. T. Pope (1988). A cookbook for using the mvc user interface paradigm in smalltalk-80. *Journal of Object-Oriented Programming 1*(3), 26–49.

Leuf, B. and W. Cunningham (2001). *The Wiki Way.* Addison Wessley Addison Wesley, Longman.

Lillehagen, F., E. Dehli, L. Fjeld, J. Krogstie, and H. D. Jørgensen (2002). Utilizing active knowledge models in an infrastructure for virtual enterprises. In L. M. Camarinha-Matos (Ed.), *Collaborative Business Ecosystems and Virtual Enterprises, IFIP TC5/WG5.5 Third Working Conference on Infrastructures for Virtual Enterprises (PRO-VE'02)*, Volume 213. Kluwer.

Miller, G., R. Beckwith, C. Fellbaum, D. Gross, and K. Miller (1990). Five papers on wordnet. *International Journal of Lexicography 3*(4), 235–312 (current version available online at ftp://ftp.cogsci.princeton.edu/pub/wordnet/5papers.ps).

Mularz, D. E. (1995). Pattern-based integration architectures. In J. O. Coplien and D. C. Schmidt (Eds.), *Pattern Languages of Program Design 1*, pp. 441–452. Reading, MA, USA: Addison-Wesley.

Odigo (2001). Company homepage. http://corp.odigo.com/.

Pelrine, J., A. Knight, and A. Cho (2001). *Mastering ENVY/Developer.* Cambridge University Press.

Pipek, V., J. Hinrichs, and V. Wulf (2003). Sharing expertise: Challanges for technical support. In M. S. Ackermann, V. Pipek, and V. Wulf (Eds.), *Sharing Expertise: Beyond Knowledge Management*, pp. 111–136. Cambridge, MA, USA: MIT Press.

Price, D. (2000). Cvs - concurrent versions system v1.11.

Prinz, W. (1999). Nessie: An awareness environment for cooperative settings. In S. Bodker, M. Kyng, and K. Schmidt (Eds.), *Proceedings of the Sixth European Conference on Computer Supported Cooperative Work*, Copenhagen, Denmark, pp. 391–410. Kluwer.

Quillian, M. R. (1968). Semantic memory. In M. e. Minsky (Ed.), *Semantic Information Processing*, pp. 227–270. MIT Press.

Rodden, T. (1996). Populating the application: A model of awareness for cooperative applications. In *Proceedings of the ACM 1996 conference on on Computer supported cooperative work*, pp. 87–96.

Rohnert, H. (1995). The proxy design pattern revisited. In J. M. Vlissides, J. O. Coplien, and N. L. Kerth (Eds.), *Pattern Languages of Program Design 2*, pp. 105–118. Reading, MA, USA: Addison-Wesley.

Rossi, G., A. Garrido, and S. Carvalho (1995). Design patterns for object-oriented hypermedia applications. In J. M. Vlissides, J. O. Coplien, and N. L. Kerth (Eds.), *Pattern Languages of Program Design 2*, pp. 177–191. Reading, MA, USA: Addison-Wesley Addison-Wesley.

Schümmer, T. (2001a). Gama-mall - shopping in communities. In L. Fiege, G. Mühl, and U. Wilhelm (Eds.), *Proceedings of the Second International Workshop on Electronic Commerce (WELCOM'01)*, Heidelberg, Germany, pp. to be published. Springer.

Schümmer, T. (2001b). 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. (2002). Enabling technologies for communities at web shops. In J. Plaice, P. G. Kropf, P. Schulthess, and J. Slonim (Eds.), *Proceedings of the 4th International Workshop on Distributed Communities on the Web (DCW2002)*, Number 2468 in Lecture Notes in Computer Science, Sydney, Australia, pp. 253 – 265. Springer.

Schümmer, T. and J. M. Haake (2001). Supporting distributed software development by modes of collaboration. In *Proceedings of ECSCW 2001*, Bonn.

Schümmer, T. and J. Schümmer (2001). Support for distributed teams in extreme programming. In G. Succi and M. Marchesi (Eds.), *eXtreme Programming Examined*. Addison Wesley.

Schümmer, T., J. Schümmer, and C. Schuckmann (2001). Coast - an open source framework to build synchronous groupware with smalltalk. Technical report, OpenCoast Development Group.

Sohlenkamp, M., W. Prinz, and L. Fuchs (2000). Poliawac – design and evaluation of an awareness enhanced groupware client. *AI and Society – Special Issue on CSCW 14*, 31–47.

Ter Hofte, G. H., R. Otte, and S. van der Gaast (1997). Supporting telepointing in the mesh groupware platform: Design issues.

Underwood, G. M., P. P. Maglio, and R. Barrett (1998). User centered push for timely information delivery. In *Proceedings of the Seventh International World Wide Web Conference (WWW7)*, Brisbane, Australia, pp. online at: http://www7.scu.edu.au/programme/fullpapers/1894/com1894.htm.

Wolf, H. and K. Froitzheim (1998). User space meets document space. In *7th International Conference on the World Wide Web*, Brisbane, pp. 710–712.