Institut für Informatik
der Technischen Universität München

Building Group Awareness in Distributed Software Development Projects

Rafal Kobylinski

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender	Univ.-Prof. Dr. Christoph Zenger
Prüfer der Dissertation:	
1.	Univ.-Prof. Bernd Brügge, Ph.D.
2.	Univ.-Prof. Prof. Dr. Johann Schlichter

Die Dissertation wurde am 01.12.2004 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 06.05.2005 angenommen.

Acknowledgments

I would like to express my gratitude to all the people who supported this work, including Prof. Bernd Brügge, Ph.D. (supervisor and principal reviewer), Prof. Dr. Johann Schlichter (second reviewer), Allen Duitoit, Ph.D. (voluntary coaching), Martin Ott (ABX server programming), Martin Pittenauer (ABX plug-ins and testbed programming), Dominik Wagner (ABX client programming), Oliver Creighton and Tobias Klüpfel (ABX pilot users), the participants of the ARENA project (ABX prospective experiment subjects), and the countless others, who provided inspiration, feedback and encouragement.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Communication is a key issue in software development projects that involve a large number of participants. Historically, Brooks was the first to provide anecdotal evidence for the critical role of communication in his experience report from the OS/360 development project in mid-seventies [14]. Today, this assumption is supported by a large number of experiments, including field studies such as those conducted by Curtis et al. [20] or Kraut and Streeter [46].

In distributed software development projects, communication becomes even more important as the participants lack physical proximity [32]. Communication breakdowns, such as misunderstandings or omissions become more frequent and harder to address. While these so-called *information pathologies* are known to occur in both distributed and not distributed organizations [70], they seem to be more frequent and more severe when *rich communication media* such as formal and informal face-to-face communication is omitted due to physical separation [22].

Current research indicates that modern communication tools such as mobile and stationary phones, email, or video conferencing equipment cannot fully substitute physical proximity. Dutoit et al. [28] provide evidence that despite modern communication technology, travel remains a crucial component of any communication strategy in distributed software development projects. The reasons are twofold. First, communication mediated by means of technology is best suited for data exchange and fails if personal relationships need to be built first [21]. Second, project participants are often completely unaware of the activities conducted at a remote site, while they are aware of the activities at their own site due to a wide range of cues they

receive by just "being there".

These findings are consistent with observations as well as informal interviews with participants, coaches and instructors of the distributed software engineering project courses conducted at the chair for Applied Software Engineering between the years 1997 and 2002 (JAMES, PAID, STARS and TRAMP) [17]. While the course participants had access to state of the art communication tools, there was a need to build relationships with remote groups.

Recent research in the CSCW field has provided some evidence that systems that improve awareness can alleviate some of the issues introduced by physical distribution and improve the overall usability of solutions for distributed work (e.g. Gutwin and Greenberg [34]).

Awareness related research reveals that most existing awareness systems are problem domain agnostic (e.g. [26], [11], [49], [83]) and have been deployed only in research environments with a limited number of users. Problem domain agnostic awareness systems which have been widely adopted outside research laboratories fall into the category of instant messaging systems (such as AOL Instant Messenger [6]).

Some problem domain specific systems exist, and those that have been developed are tightly coupled to a single application environment for a given problem domain. Moreover, quite often the problem domain chosen is artificial, resembling a game or a simulation rather than an actual work environment (e.g. collaborative *simulated* pipe welding [34]).

Problem domain specific awareness systems for distributed development projects are limited to a restricted application environment. One example is the TUKAN environment for SMALLTALK developers developed by Schümmer [72]. While TUKAN provides a good example on how developers can benefit from improved awareness, it also requires that the developers use it as a replacement for their development environment.

We believe that most developers will not switch to a completely new set of tools just to gain the claimed benefits of awareness. Thus we follow an approach to implement awareness that enhances and complements existing development tools.

Many of the existing systems have not been evaluated experimentally. To gain insight into the impact of awareness systems on communication and outcome of distributed projects, there is a need for comprehensive experimentation in the context of real software development projects.

1.2 Goal and Approach

The goal of this dissertation is to assess the impact and usability of a domain specific awareness system for distributed software engineering projects. To accomplish this goal, we follow an experimental approach. We first analyze the requirements for awareness systems. Then we design an architecture for awareness systems that can coexist with existing development tool. Finally we realize a prototype system instantiating this architecture and evaluate the prototype in the context of distributed projects.

An awareness system needs to support the collection of awareness events from a wide range of tools. For this purpose, the architecture needs to provide an open interface based on protocols that allow easy integration of development and communication tools. Rather than using new protocols designed from scratch, our prototype builds on protocols that have emerged in recent years, such as XML-RPC [77] and SOAP [80].

To limit the information load on the users, the awareness system has to provide a means to assess the relevance of incoming events for individual users. How to efficiently query dynamic event data streams (rather than static, historical event data stored in a conventional database) is an open research problem [8] and out of scope for this work. Rather, the focus in this dissertation is to describe a usable method for specifying relevance weighted interests. Work on relevance assessment in awareness systems that has been done before by Bürger [18] is used here as a starting point.

The awareness system needs to support different ways of disseminating the rated awareness events and presenting them to the users of the system. Similar to the collection of awareness events, the dissemination also needs to support a wide range of dissemination means and presentation devices. While it is often easiest to provide users with a separate tool that receives and presents the rated awareness events, some applications are extensible enough to allow presentation of awareness events in their own context. Moreover, some users might prefer to use ambient presentation systems [83] rather then traditional desktop applications to reduce clutter on their computer desktop. Thus the architecture needs to be extensible as well.

The design of an architecture based on these requirements is only a first step of this work. The second objective of this dissertation was to build a working prototype of an awareness system that fits an existing development and communication environment. A working prototype allows experiments that can help to understand the impact of an awareness system on software development projects. It is the only means to validate that the proposed architecture can be used as the foundation for a usable and effective system.

With the prototype system called AWARENESS BUILDER, we conducted

experiments with students as human subjects to better understand the impact of improved awareness in software development projects and provide us with important feedback on how we need to improve our architecture.

Several distributed software engineering project courses formed the context for the empirical evaluation. The students in these classes have a realistic development experience due to interaction with an industry client and to the requirement of designing and building a large scale prototype system for this client.

During each course, students have access to a state of the art communication and development infrastructure, which includes tools such as LOTUS NOTES DOMINO web-based groupware, video conferencing, CVS software configuration management and tools for editing of artifacts such as UML models. An important goal of the AWARENESS BUILDER prototype was to allow integration of awareness into this existing environment.

We performed two studies. The first study was retrospective using historical data from past distributed development project courses (PAID, STARS [27] and TRAMP) [17]. The goal of this study was to calibrate the prototype and to improve its overall usability and usefulness.

The second study was prospective and performed in a running distributed project course (ARENA [2]). While the first study was limited by the amount and the quality of the available historical log files, the second study benefited from the availability of the project participants for feedback elicitation.

1.3 Contributions

The contributions of this dissertation are as follows:

- It is shown that awareness systems share simple functional principles, but differ in the way they interact with the user. A matrix consisting of four quadrants is established and used to classify a selection of existing awareness systems in terms of presentation technology and interaction principles used. The common functional principles are formalized and presented in detail as requirements for awareness systems.

- An awareness architecture for distributed software development projects is presented along with a proof-of-concept implementation called AWARENESS BUILDER. Contrary to other efforts to implement awareness that can be found in the literature (e.g. TUKAN [72]), AWARENESS BUILDER does not require the replacement of the existing set of communication and development tools. Rather, it is an extensible external tool that can be configured to provide awareness with little changes to the

existing environment. Focusing on functional principles, AWARENESS BUILDER provides a graphical but simple user interface.

- An quantitative evaluation method for awareness systems is presented. Most publications on awareness systems either lack evaluation results or base their evaluation results on qualitative methods only. The method presented in this dissertation assumes that changes to project artifacts are tracked and that project communication is recorded. The method then allows to observe the flow of information that an awareness system created during the course of a project. It can also observe the flow of information that an awareness system *would have* created on past projects that did not use an awareness system. In addition, repeated application of the method on the same project allow to fine-tune an awareness system.

- A quantitative retrospective evaluation of AWARENESS BUILDER using three past software development projects with 78 – 121 participants and 25,071 – 231,601 lines of code. Evidence is found that relevance assessment is needed because of the volume of awareness information that is created in projects of that type. The evaluation also shows that relevance assessment based on a small set of simple rules can reduce the volume of information by 98.5% – 99.2%.

- A quantitative prospective evaluation of AWARENESS BUILDER using an ongoing software development project with 37 participant and 16,546 lines of code. The quantitative evaluation provides evidence that users will increase the volume of information, if given the chance to: in our case by 52.8%. Thus the overall reduction of the volume of information in the ongoing project was only 94.7%.

- A qualitative evaluation of AWARENESS BUILDER in an ongoing software development project (prospective evaluation). User observation, informal discussions and feedback via a questionnaire revealed "soft" facts about AWARENESS BUILDER. Users have reported that the tool has triggered communication between them and fellow project participants and that the tool prevented them from doing conflicting work on an artifact.

1.4 Thesis structure

Chapter 2 introduces the basic concepts of awareness and surveys the various approaches that can be found in the literature. It lays the groundwork for

the proposed awareness architecture.

Chapter 3 describes the distributed software engineering project courses used for requirements elicitation and for the experiments.

Chapter 4 describes the requirements and features that awareness systems need to support in order to be usable and to fit into the exisiting application environments.

Chapter 5 describes the architecture derived from the requirements presented as well as AWARENESS BUILDER, a working prototype implementation based on this architecture. It includes a discussion of the level of integration that was reached with the existing infrastructure.

Chapter 6 discusses the results of two empirical studies performed. It includes the description of the methods and metrics used in those studies. Moreover, a discussion of the metrics, logging mechanisms and tools needed to perform the studies and obtain meaningful results can be found here.

Chapter 7 outlines future research and development directions.

Chapter 2

Group Awareness Model

In this chapter, we define the concept of group awareness (Section 2.1), summarize the claimed benefits associated with awareness systems (Section 2.2), and survey a selected number of awareness systems (Section 2.3). We close this chapter with a discussion of some typical issues raised by awareness systems (Section 2.4) and a summary (Section 2.5).

2.1 Towards a Definition of Awareness

The term *awareness* has a broad range of meanings. The most common definition — one that can be found in a dictionaries (e.g. Merriam-Webster-Online [5]) – is that awareness is the state of having perception and/or knowledge of something. Consequently, in medicine and psychology awareness is often synonmous to consciousness.

A definition more specific to the context of collaborative work relates awareness to the working environment, e.g.

Definition 2.1 (Awareness – Dourish and Belotti [25]) *(...) awareness is an understanding of the activities of others, which provides a context for your own activity.*

This definition implies a group of people working together, and in fact this kind of awareness is often referred to as group awareness in the CSCW research community and the terms *awareness* and *group awareness* are used interchangeably[1].

[1]Group awareness in this sense is not related to the controversial "Large Group Awareness Trainings" or LGATs, which emerged in the 1960s and were part psychotherapy, part sprituality, and part business [47]

Usually, in colocated work groups, awareness is acquired automatically because of the wide range of audiovisual cues available for conscious and subconscious interpretation.

Studies by Kraut et al. [45] show that the communication frequency suffers a logarithmic decline with distance between potential communicators. The communication frequency appears to drop sharply at a distance corresponding to the length of a typical hallway:

Kraut et al. where able to show that researchers who had offices next to each other had approximately twice as much communication as those whose offices were on the same floor, but at a greater distance. A distance greater than 30 meters between offices or offices on different floors was almost as bad for communication as distances from offices in different buildings, with communication reduced by 80%. This communication decline was observed not only for face-to-face communication, but also for phone calls as well as electronic mail messages.

This leads to the following definitions:

Definition 2.2 (Colocated work group) *A work group is called colocated if and only if all group members have offices on the same hallway (that is less than 30 meters apart).*

Definition 2.3 (Distributed work group) *Any work group that is not colocated is called distributed.*

We believe that all distributed work groups have very limited means of acquiring awareness and need some sort of support.

The Awareness definition from Dourish and Belotti [25] is quite vague. In fact, according to that definition, a lot of information can be considered awareness information. A list of eleven information elements that can be considered relevant for group awareness was proposed by Gutwin and Greenberg [33]:

Presence	Who is participating in the activity?
Location	Where are they working?
Activity Level	How active are they in the workplace?
Actions	What are they doing?
	What are their current activities and tasks?
Intentions	What will they do next?
	Where will they be?
Changes	What changes are they making, and where?
Objects	What objects are they using?
Extents	What can they see? How far can they reach?
Abilities	What can they do?
Sphere of Influence	Where can they make changes?
Expectations	What do they need me to do next?

All those information elements make the common assumption that there is a shared workspace populated by people and entities[2].

The work environment can be modelled with a class `Workspace` that is an aggregation of `Person` and `Entity` (Figure 2.1). There is no need to make any assumptions about people and entities other than instances of `Person` need to be able to send messages to instances of `Entity` (therefore our model includes an association between these two classes). As people can usually work on several entities at the same time and also can work either single-handed or collaboratively on one entity, the multiplicity of this association is set to many-to-many.

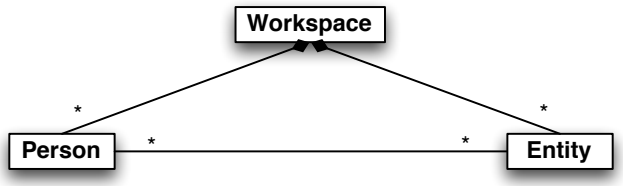

Figure 2.1: The workspace consists of people and entites.

For every single activity that occurs in the real world work environment, an instance of `Person` needs to invoke an operation on an instance of `Entity` (by means of sending it a message) in our model. For example, if — in the real world — a person enters a room, our model needs to reflect this activity by having an instance of `Person` send a message `enter()` to an instance of

[2]we prefer the term *entity* rather than *object* to avoid confusion with object-oriented terminology used in our models.

`Entity` (or more precisely, a specific subclass of `Entity`). Later, when — again in the real world — that person decides to leave the room, than the same instance of `Person` has to send a message `leave()` to that instance of `Entity`. This type of dynamic behaviour is illustrated in Figure 2.2.

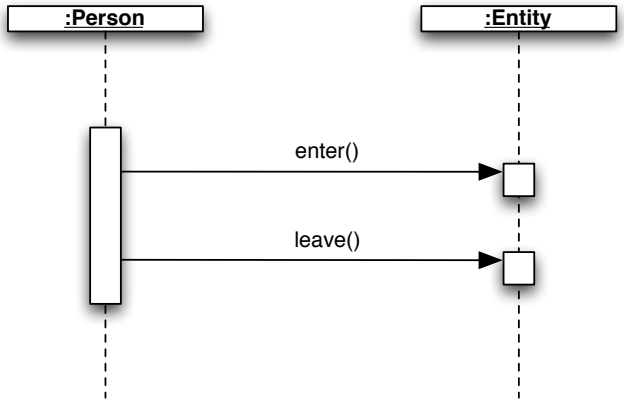

Figure 2.2: The simple workspace model presented in this section maps activities in the real world to message invocations on `Entity` instances.

In a traditional real world workspace, it is difficult to monitor all interesting activities without being there in person due to the lack of sufficient instrumentation or sensors. To allow observers of such a pure or *non-instrumented* workspace to answer the above questions stated by Gutwin and Greenberg, some monitoring mechanism has to be added to the workspace and transform it into an *instrumented* workspace. This to the definition of the instrumented workspace as follows:

Definition 2.4 (Instrumented Workspace) *An instrumented workspace consists of people (who are the active elements of the workspace), entities (which are the passive elements of the workspace), and a monitoring mechanism that allows to track the people's actions on entities.*

In our model, we assume an Instrumented Workspace with a monitoring mechanism that creates `AwarenessEvent` instances each time an activity occurs. To provide for actual tracking, these instances need to store at least the message sent, the invoker (a `Person`) as well as the receiver (an `Entity`) of the message, and a time stamp (Figure 2.3).

With these `AwarenessEvents`, it becomes possible to answer some of the above questions, if we assume that these objects are not immediately deleted but rather stored and can be later retrieved automatically. Questions about

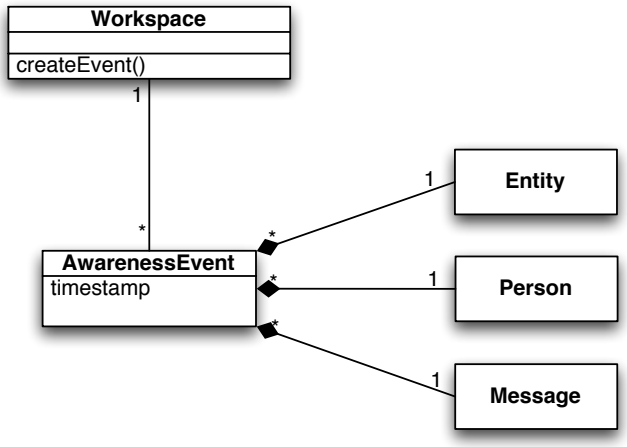

Figure 2.3: A monitoring mechanism needs to create `AwarenessEvent` instances for each message invocation.

presence, actions, and entities can be answered directly: one has to retrieve the respective objects (`Entity`, `Person` or `Message`) associated with recently stored `AwarenessEvent` objects.

In this model *Location* is just a special case: questions about location can be answered, if we assume that a `Location` is a special type of `Entity`, e.g. a room, a floor or a building (this is implied in the example presented in Figure 2.2) and that access to it can be monitored. Questions about activity level can be answered by looking at the frequency new `AwarenessEvent` related to a certain activity are created.

Therefore we will call the components of the `AwarenessEvent` the core elements of group awareness.

Definition 2.5 (Core Elements of Group Awareness) *The core elements of awareness are presence, action and entity awareness. They can be acquired by monitoring* `AwarenessEvents` *in an Instrumented Workspace.*

The other questions (e.g. Questions about Intensions: What will they do next? Where will they be?) require context knowledge and the answers cannot be derived from monitoring the activities alone. As those answers are usually not obvious in both collocated as well as distributed settings, we will focus on the Core Elements of Group Awareness in this dissertation.

We conclude that distributed work group need mechanisms for awareness acquisition. In terms of our workspace model, a monitoring mechanism is needed to create `AwarenessEvent` objects and a complementary distribution

mechanism is needed to forward them to interested observers. We will call such a system that provides these mechanisms an Awareness System:

Definition 2.6 (Awareness System) *An Awareness System provides monitoring and* `AwarenessEvent` *distribution services to distributed work groups.*

We will discuss the requirements for such an Awareness System and the design space for the monitoring mechanism in more detail in Chapter 4. Assuming an Instrumented Workspace with a functional Awareness System in place, we now define Awareness as follows:

Definition 2.7 (Awareness) *Given a set E of all* `AwarenessEvent` *objects of a given* `Workspace` *and an observer o of type* `Person`*,* Awareness A_o *is a subset of E:* $A_o \subset E$*.*

A_o is the set of `AwarenessEvent` objects and thus those activities in the Workspace, that the observer o is aware of, by means of the Awareness System's event distribution services.

2.2 Claimed Benefits

It is widely accepted that group awareness is an important communication facilitator, especially in a distributed setting.

Dourish and Bly claim that awareness leads to *informal interactions, spontaneous connections, and the development of shared cultures.* They further state that these are *important aspects of maintaining working relationships which are denied to groups distributed across multiple sites* [26].

They provide experimental evidence for their claims with the PORTHOLES system (described in detail in Section 2.3.2), which was tested with a group of 22 researchers at Xerox PARC Palo Alto and Xerox EuroPARC in Cambridge. The results, based on user observation and structured feedback, suggest that *awareness across distance (...) can lead positively toward communications and interactions, and perhaps most importantly, that it can contribute to a shared sense of community* [26].

Another experiment at Xerox involving a laboratory in Palo Alto and in Portland revealed a real need for awareness in distributed settings. While the MEDIA SPACES system used in that experiment was not specifically designed as an awareness system, Bly et. al claim that *the use of the media space for (...) awareness was perhaps its most powerful use* [11]. The MEDIA SPACES system was evaluated in a three year case study involving a group of approximately 20 researchers.

Quite often, awareness systems are presented without any empirical evidence being provided to support the claims of the authors (e.g. MIT's AMBIENT ROOM [83]). We believe that this is not because those benefits do not exist, but because they are difficult and expensive to measure.

Therefore we think that any claims about awareness systems need to be supported by careful experimentation. In this work, we evaluate the reported ideas empirically in the context of a distributed project. The experimental approach and the results are described in Chapter 6.

2.3 Survey of Related Systems

2.3.1 Overview

Early awareness research concentrated on video-based systems. The MEDIA SPACES [11] system that linked the research laboratories in Palo Alto, California and Portland, Oregon provided awareness with a live video view of the remote collaborators' office activity. It's excessive use of costly bandwith led to the development of PORTHOLES [26], which required less network bandwidth by using occasional video snapshots instead of a live video feed.

Later, systems used processed snapshots instead of video (e.g. PIAZZA [40]), and line drawings (e.g. PEEPHOLES [31]) to further reduce the network bandwidth. Some researchers started to experiment with symbolic representations (e.g. [58]), or virtual worlds with avatars (e.g. FORUM [44]). Even though bandwith limitations are not as much of a problem as they used to be (at least in stationary environments), symbolic representations tend to be better at preserving privacy and can be also better filtered to limit the information load (the next section covers these issues in more detail). Moreover, symbolic representations can be easier to interpret for the user (e.g. reading a user's name might be sometimes easier than trying identify who the person on a low-resolution picture is).

Another category of awareness systems are instant messaging systems, which combine a notion of presence in a virtual space with simple means of computer mediated communication, often called "chat". Those systems have become widely popular in recent years and share the Internet standard Internet Relay Chat (IRC) [55] as their common predecessor.

Instant messaging systems such as AOL INSTANT MESSENGER [6], ICQ [39], JABBER [42], MSN MESSENGER [51] or YAHOO! MESSENGER [84] all provide a basic awareness service by indicating the presence of coworkers — if the coworkers are logged into the instant messaging system. Coincidentally, these system usually indicate presence in a symbolic way, providing either only a

name or a combination of a name with an image from an electronic address book (e.g. ICHAT [7] — Figure 2.4)

Figure 2.4: ICHAT indicates that user "Oliver Creighton" and "Ullrich Bauer" are available while "Dominik Wagner" is not (name grayed out)

The XWHO [49] system that also indicates the presence of coworkers and includes a chat-like functionality. Unlike the instant messaging systems mentioned sofar, XWHO does not require a centralized controlling entity. The XWHO system consists of two local components: the XWHO server captures the information about who is currently logged in and shares this information with all XWHO servers on the network, while the client displays the information known to the local XWHO server to the user in a window. Thus, in addition to indicating presence, this system also indicates the physical location, by showing at which workstation the remote coworker is sitting.

The BABBLE [29] is a client-server multi-user communication system that provides awareness information about a user's activity level when the user is interacting with the BABBLE client. BABBLE includes persistent server-based storage for all conversations and focuses on multilateral rather than just bilateral conversations.

While early systems such as MEDIA SPACES and PORTHOLES used rich media to capture and reproduce as much information about a remote work environment as possible, these recent systems use simpler media and a more symbolic approach. Most of recent systems still use a traditional display — the display of a computer — competing for display space with the applications the user uses to complete his work. This may be not desirable as staying "aware" is often considered a background activity, and should not conflict with the foreground activities, but rather provide additional information through other channels.

Systems like AMBIENTROOM [83] try to resolve this issue by moving the awareness display away from the workstation to the ambient environment. In such systems the information is not presented on the computer screen, but directly in the physical environment and available through any of the five senses.

Such ambient interfaces can also provide ways to interact with the system. For example, Huang et al. [38] from the Accenture Technology Labs have deployed a large display in a public area of their office. Pieces of awareness information are randomly displayed for 15 seconds at a time, resembling the experience of walking by a newspaper dispenser and seeing the headlines. The system allows access to additional information by swiping the personal ID badge at an integrated badge reader — if additional information for the currently displayed item is available, the system will email this information to the owner of the ID badge[3].

As awareness information can be presented either in a literal or symbolic way and through both, conventional desktop interfaces as well as ambient interfaces, we can use those two dimensions of presentation styles to build the awareness presentation matrix to illuminate the differences between awareness systems (Figure 2.5).

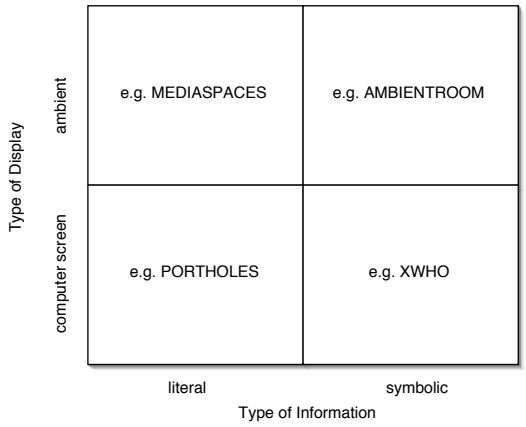

Figure 2.5: Awareness presentation matrix

The lower-left quadrant of the awareness presentation matrix includes systems that provide literal awareness on computer screens. A typical example of such a system is PORTHOLES (see Section 2.3.2), which broadcast video snapshots to traditional desktop environments. The upper-left quadrant includes systems that provide literal awareness in the ambient environment.

[3]the email has to be accessed traditionally.

MEDIA SPACES (see Section 2.3.3) also broadcasts video, but displays it on dedicated monitors rather than on the desktop.

The upper-right quadrant includes systems that provide symbolic awareness in the ambient environment. A comprehensive example is AMBIENTROOM (see Section 2.3.4), which combines several experimental ambient displays to present symbolic awareness data. Finally, the lower-right quadrant includes systems that use traditional desktops to provide symbolic awareness. XWHO (see Section 2.3.5), is a typical example.

The awareness presentation matrix is a simplified representation implied by a class hierarchy for awareness systems (Figure 2.6). This class hierarchy involves two root classes, `UserInterface` and `AwarenessData`, that correspond to the two dimensions of the matrix. Each of these two root classes has two subclasses, that correspond to the two values that each dimension supports. At the bottom of the class hierarchy, there are four leaf classes that correspond to the four quadrants of the matrix: `GraphicalLiteral`, `GraphicalSymbolic`, `AmbientLiteral` and `AmbientSymbolic`.

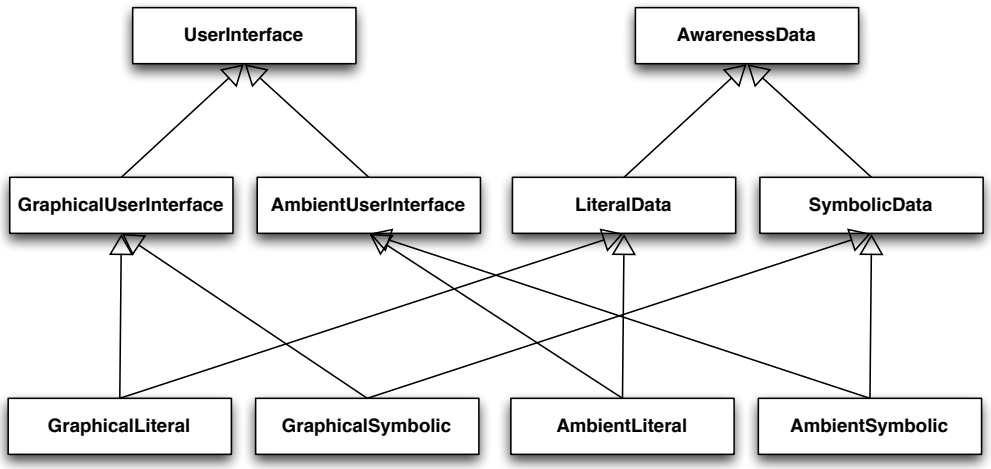

Figure 2.6: The awareness presentation matrix is implied by the leaf-classes of a class hierarchy for awareness systems.

Next, for each of these classes, we describe one representative instance, namely an exisiting awareness system, in detail.

2.3.2 GraphicLiteral Awareness System: Portholes

PORTHOLES is an example of a system which presents information from the remote site(s) in a graphic-literal way. It uses a traditional desktop user in-

terface that incorporates windows and GUI elements such as buttons, menus and scrollbar, which can be controlled with a keyboard and a mouse (Figure 2.7). It provides an overview of one's community by presenting a matrix of still video images. These images are updated periodically (e.g. every 5 minutes).

Figure 2.7: PORTHOLES client window (screenshot from [26], used with permission [24])

The PORTHOLES architecture consists of two distinct sets of components: a set of servers, which deal with information acquisition and distribution and a set of clients, which deal with the presentation of the information to the user (Figure 2.8). Each server is connected to an image capture facility.

PORTHOLES servers are not tied to a single client but provide an open client interface — thus, different types of clients are possible. Consequently, three different clients have been implemented: PVC, EDISON and VIEWMASTER.

PVC is the simplest client, which presents a matrix of still video images and updates them periodically. The user interface is presented in Figure 2.7. The user can influence the selection of displayed images either by providing a configuration file or by selecting them interactively by means of a menu.

In addition, EDISON allows to record audio messages and to listen to messages recorded by others. Thus, rather than providing only awareness, it integrates a limited form of communication. Both PVC and EDISON are integrated with the email systems: clicking on an image brings up a dialog

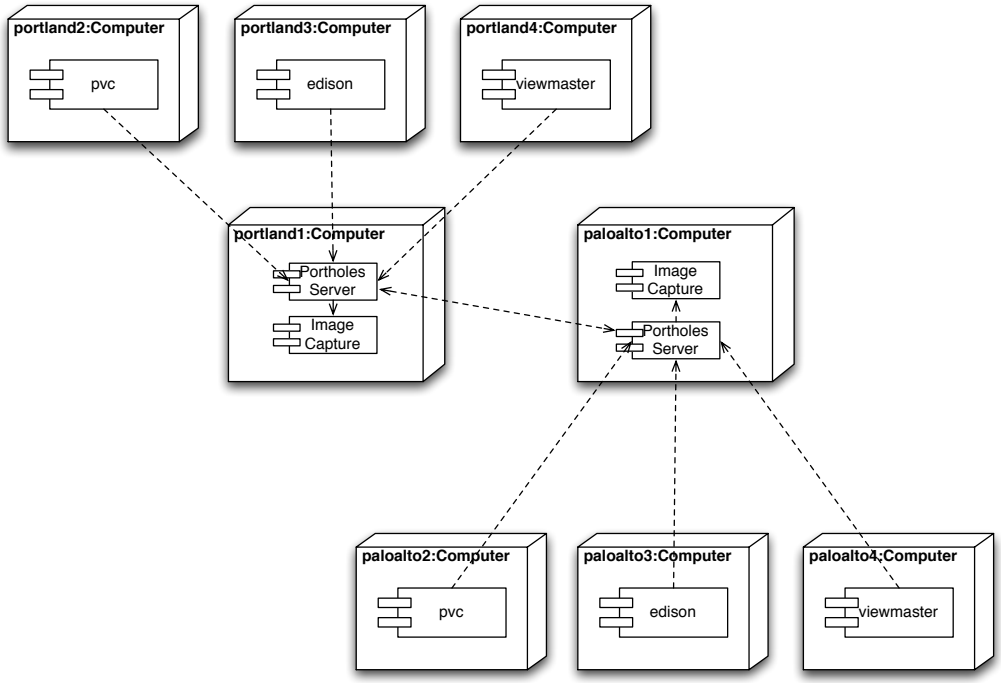

Figure 2.8: PORTHOLES architecture (adapted from [26]) with two servers and six clients for viewing awareness information

displaying information about the image and action buttons allowing send an email to the person associated with the particular image.

The third client, VIEWMASTER, is a client meant for public use. A viewer that can view only video images marked as public. Being a research prototype, PORTHOLES lacks any more sophisticated means of access control other than this distinction between public and non-public images.

By providing periodically updated still video images PORTHOLES awareness support is limited. The PORTHOLES user can usually identify the people, the places and also some real-world objects when looking on the PORTHOLES client window. However, some objects that are being worked on, may not be visible. Also, in settings where people tend to work on non-physical, "virtual" objects this information is typically always hidden. Most of the activities will remain hidden as well. The notion of history is not supported in PORTHOLES.

2.3.3 AmbientLiteral Awareness System: Media Spaces

The Xerox Palo Alto Research Center (PARC) MEDIA SPACES system [11] provided live video links between two parts of a research laboratory that was split between Palo Alto, California and Portland, Oregon. It is an example of a system that presents information from a remote site in a literal way. It does not use use the computer desktop to present that information, and can be therefore classified as an instance of `AmbientLiteral` class of awareness systems.

While MEDIA SPACES was inspired by traditional visual communication systems such as video conferencing systems and picture phones, it differed from such systems, because of its passive nature.

The experiment began 1985 and lasted for three years. Initially , the Palo Alto and Portland labs were linked together by a low-bandwith link (initially 9 kbs, later 56 kbp) which allowed for cross-site file sharing and electronic mail services. Additionally, speaker phones with phone conference support were provided. Later[4], an always-on two-way video link was created.

Each site had, in addition to private offices for the group members, a common informal communication area, where a camera, a monitor and a speaker phone dedicated for the video link were placed. The camera was connected to compression equipment which allowed to compress the video to 56 kbps. The link was left open all day, allowing anyone in one of these two common areas to see and hear anything that was going on in the remote common area.

While Bly, Harrison and Irwin expected the link to be used as a inter-site meeting tool, they observed that it was primarly used for informal communication and awareness acquisition.

After gaining some experience with the initial video setup, Bly et al. extended it to include individual offices. Cameras, monitors and microphones were installed in each office and connected to central, computer-controlled crossbar switch. Thus, in addition to the connection between the common areas made possible by the initial video setup, direct connections between private offices as well as connections between a private office and the common area became possible. While anyone had access to a controller application for the switch and could change the configuration of it anytime at will (and changes reportedly occurred many times a day), privacy was dealt with mostly by turning off the microphone rather than the camera.

Bly, Harrison and Irwin report also that the primary use of MEDIA SPACES was to acquire information about presence of others, in order to specifically

[4]No specific date is mentioned in [11]

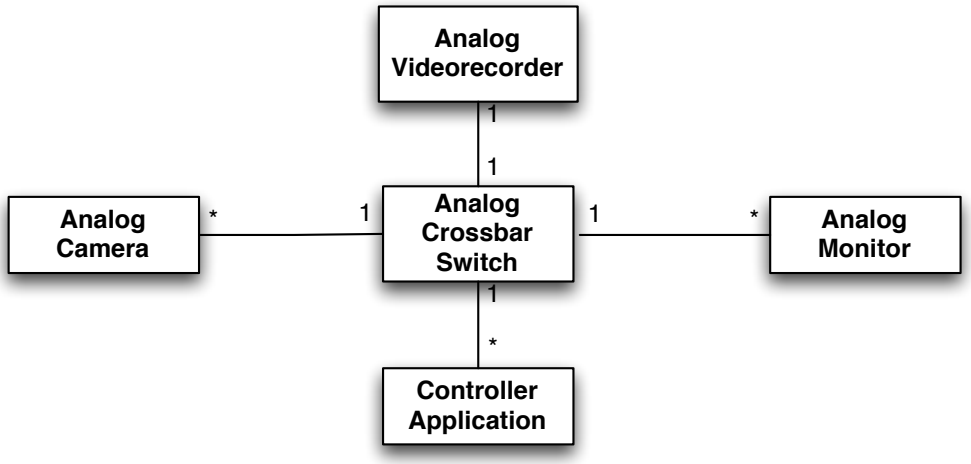

Figure 2.9: MEDIA SPACES architecture (adapted from [11]) — a number of video cameras and video monitors can be connected to a singleton crossbar switch.

participate in chance encounters and to locate colleagues. Somewhat surprisingly (for the system designers), despite heavy use of the system, planned video phone conversations were less frequent than initially expected.

By providing a fixed, two-way audiovisual link, MEDIA SPACES supports audiovisual monitoring, although it shares the limitations of all literal awareness systems, in particular not being able to figure out what happens with "virtual" entities (rather than real-world entities). History was supported by connected recording and playback devices such as video recorders and laser disk players.

It seems that the use of separate, non-computer displays for monitoring remote locations was a result of technological limitations rather than the result of a deliberate descision. Nevertheless, the authors mention that the primary use of MEDIA SPACES was indeed the acquisition of *awareness* — we assume that this would had been much harder, if the monitoring had to be done via windows on the usually already very cluttered computer displays.

2.3.4 AmbientSymbolic Awareness System: Ambientroom

AMBIENTROOM has been developed by the MIT Media Lab as *a new kind of awareness support technology including, but not limited to, the "awareness" of people's activities* [83]. It is an example for an awareness system that

that can be easiliy mapped to a numerical (one-dimensional) value. The creation frequency of a certain kind of `AwarenessEvent` objects can be mapped to such a value — i.e. the higher the number of individuals entering a given room the higher the value etc. However, such a mapping introduces a loss of awareness information as there is no way to find out who exactly is in a room etc. Thus, the analog nature of these displays suggests that they should be rather used to complement traditional desktop displays rather than a sole source of awareness information.

Taking the information overload issue (defined later in this chapter) into account, the AMBIENTROOM prototype as a whole is probably too cluttered with ambient display technology to be usable in a non-research environment — still, nothing prevents the usage of any of the three display technologies in a working environment that does not exectly match the AMBIENTROOM example. Unfortunately, the authors do not present any experimental results of actual use of the system.

However, other, less intrusive proposals for systems that fit into the same category exist that are supplemented with case studies that provide evidence that ambient displays can fit in a working environment.

An early example is Jeremijenko's "Dangling String" installation at Xerox PARC that mapped the traffic intensity of the laboratory's LAN to movements of a physical plastic string attached to the ceiling [81]. The string was placed in a unused corner of the lab and could be seen and heard from many cubicles while reportedly not being disturbing.

"Informative Art" [73] is a recent proposal to use state-of the-art display technology such as large plasma displays for dynamic visualizations of awareness information like e-mail traffic or activity level in a room and a presentation style for the information that is inspired by Modern Art painters[5] (Figure 2.11).

A questionnaire survey involving 40 students provided evidence that the majority perceives "Informative Art" as an enjoyable and natural part of the surroundings. Users that did not receive any explanation before the experiment were perceiving the system as art only, and also as a natural part of the surroundings.

2.3.5 GraphicalSymbolic Awareness System: xwho

XWHO is an awareness system [49] for a family of workstations that allows users and administrators to stay informed about who is using the workstation. It requires functionality that is provided by most multi-user operating

[5] e.g. Piet Mondrian, Bridget Riles, Andy Warhol and Mark Rothko

abandons traditional computer desktop user interfaces and surrounds the user with an augmented environment, capable of providing subtle, background audiovisual and symbolic cues.

Figure 2.10: The AMBIENTROOM system from the MIT Media Lab (photograph from [83], used with permission [41]) is based on a closet-like office product from Steelcase Corp.

The AMBIENTROOM system is based on a cubicle-like product from Steelcase Corp. called "Personal Harbor" (Figure 2.10). "Personal Harbor" resembles a small closet with an integrated desk, shelf and several small drawers. This mini-office has been augmented by the MIT researchers with three innovative displays: water ripples, active wallpaper and ambient sound.

Water ripples were created by a combination of a lamp and a water tank that produced rippling shadows on the ceiling of the mini-office. The frequency of these shadows could be influenced with a selenoid in the water tank — this way awareness information could be directly mapped to the water ripples on ceiling. The creators of AMBIENTROOM used this display to represent the activity of a resident hamster they had in their laboratory.

Active wallpaper was a wall in the mini-office, on which patterns of illuminated, moving patches could be projected. Depending on the level of activity in a common area equipped with movement sensors, the patches could move slower or faster, providing a cue for the activity in that remote area.

Finally, AMBIENTROOM was able to map sensors that would notice activity on a digital whiteboard located in a common area to low-volume sounds of dry-erase pens rubbing against a whiteboard.

Water ripples, active wallpaper and ambient sounds all seem to be a well suited to represent continuous awareness information: awareness information

Figure 2.11: Piet Mondrian-style abstract visualization of weather conditions (photograph from [73], used with permission [37])

systems such as Solaris, Linux or Mac OS X.

Each time a user logs in, the login is recorded in a binary file — the utmp-file. Systems that support this functionality also provide an interface for programmers to read the contents of this file. For regular users, command line tools like WHO, W and USERS display contents of that file in a human readable format.

For users on the local subnet, the RWHO command line tool is used to display the contents of remote utmp-files. This tool relies on the RWHOD server which broadcasts the local login information on the network, and collects broadcasts from other RWHOD servers to store them in a local database.

XWHO augments this functionality by providing a X WINDOW SYSTEM based graphical user interface (Figure 2.12). In addition to this change in presentation media, XWHO updates the display regularly, allows the user to define multiple presentation layouts with a configuration file and allows to observe any workstation that is reachable via TCP/IP.

The user definable layouts presents the login information in a way that

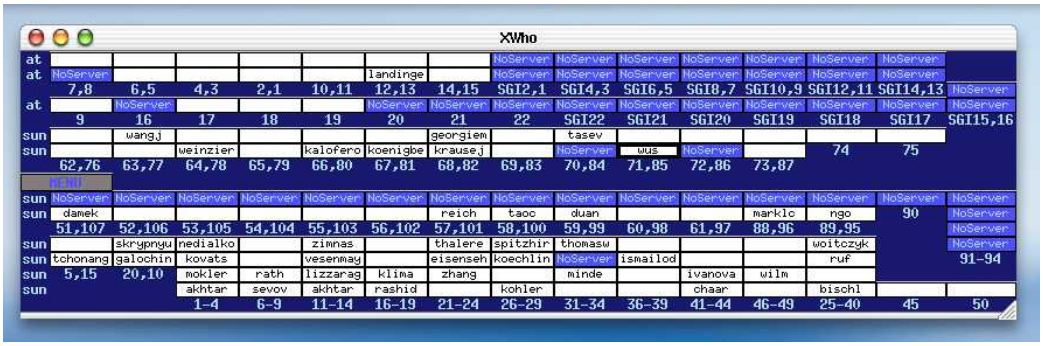

Figure 2.12: XWHO client window

resembles the physical layout of the workstations in a computing lab. Unfortunately, the layout is restricted to a rectangular matrix, which makes it hard to map certain settings on the layout. However, different layouts with different workstations (e.g. for different labs) can be defined and selected during runtime.

In terms of our model, XWHO is a monitoring mechanism for login and logout activity on workstations (Figure 2.13). There is no way to figure out what the person logged in to a remote workstation is doing, though, and there is no notion of history. XWHO provides limited support for communication, in the form of a button that allows to initiate a TALK session with the selected user (TALK is the command line based predecessor of modern chat systems).

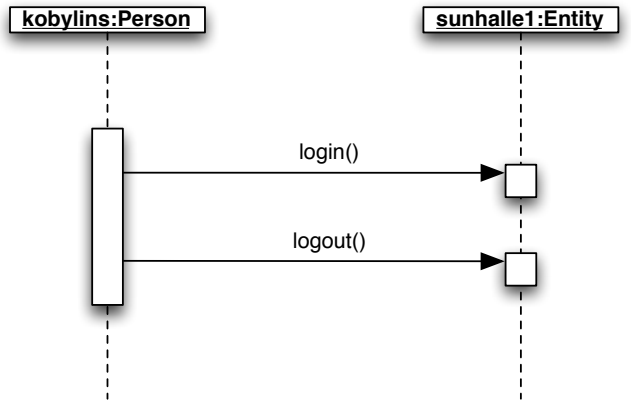

Figure 2.13: XWHO is a monitoring facility for login and logout activity on workstations.

2.3.6 A Modern Example: TOWER

The four previous sections describe prototypic examples of the four categories implied by the awareness presentation matrix. These are early examples which are, with the exception of XWHO, widely cited in the literature (e.g. as of November 2004, Google [3] reports approx. 2,990 pages when searched for the three keywords `dourish`, `bly`, and `portholes`). As stated at the beginning of this section, newer systems tend to employ more advanced techniques such as virtual worlds with avatars (symbolic representation of human beings) to present awareness information to the end user.

Still, most newer systems fit into the GraphicalSymbolic class of awareness systems and share their presentation characteristics with simpler systems such as XWHO, because they use symbolic metaphors and conventional computer displays to present awareness information. Some newer system share the characteristics of the GraphicalSymbolic and AmbientSymbolic classes, because they allow to use both traditional (i.e. GUI-based) and ambient display technologies.

An example of such a system is the TOWER — Theatre of Work — system [43][64][66][71]. The main kind of display used by the TOWER system is a computer rendered 3D world (Figure 2.14) used to represent the users of the system and their activities.

Figure 2.14: Tower 3D world: avatars on buildings represent users acting on respective documents, used with permission [65].

Ambient sensors and sensors built into shared applications ("information spaces" [43], e.g. Lotus Notes) collect events that represent the users' activities . These events are than mapped to a 3D rendering where the users are represented as avatars and the activities are represented using spacial metaphors (e.g. shared editing of a document is represented as a gathering

of avatars).

The TOWER system generates the 3D world dynamically from the shared information and the objects used by the users of the system using a set of rules. Thus, the 3D world changes constantly as artifacts are created, modified and deleted. The rule-based approach allows to tailor the mapping between the artefacts and their 3D representations to a particular application domain such that the distances between objects in the 3D scenery map to topical distances between the artefacts they represent.

In addition to the 3D presentation, the TOWER system supports ambient displays, ranging from projections of the 3D world in the office environment to "flowers" that move to signal certain events.

The TOWER system is a comprehensive example of a modern awareness system that uses advanced visualization techniques to present awareness information in 3D.

2.4 Issues with Awareness Systems

2.4.1 Privacy Considerations

Awareness systems tend to raise privacy concerns. Most people dislike the idea of being secretly observed, eavesdropped, or monitored by others, especially, if they cannot see the observers. Unfortunately, literal awareness systems, such as those that use always-on video like MEDIA SPACES, are built deliberately for the purpose of observing others without "being there". One common technique to counter these concerns is to manipulate the images with filters (Figure 2.15).

Recent studies suggest that video images can carry awareness information in spite of the application of filters like blur and pixelize, while safeguarding the privacy of the filmed people.

The study by Boyle, Edwards and Greenberg [13] has shown that the protection of privacy decreases and the number of identifiable awareness cues increases as the fidelity of the video image increases. To find a filtration level that would provide basic awareness while safeguarding privacy, the researchers pre-produced a series of videos and applied blur and pixelize filters at different filter levels to them.

They asked 20 subjects to view these videos and answer questions in a questionnaire about awareness cues like number of people, posture, gender, objects, actor activity and availability. The subjects were also asked questions about how well their privacy would be protected if they themselves would had been filmed.

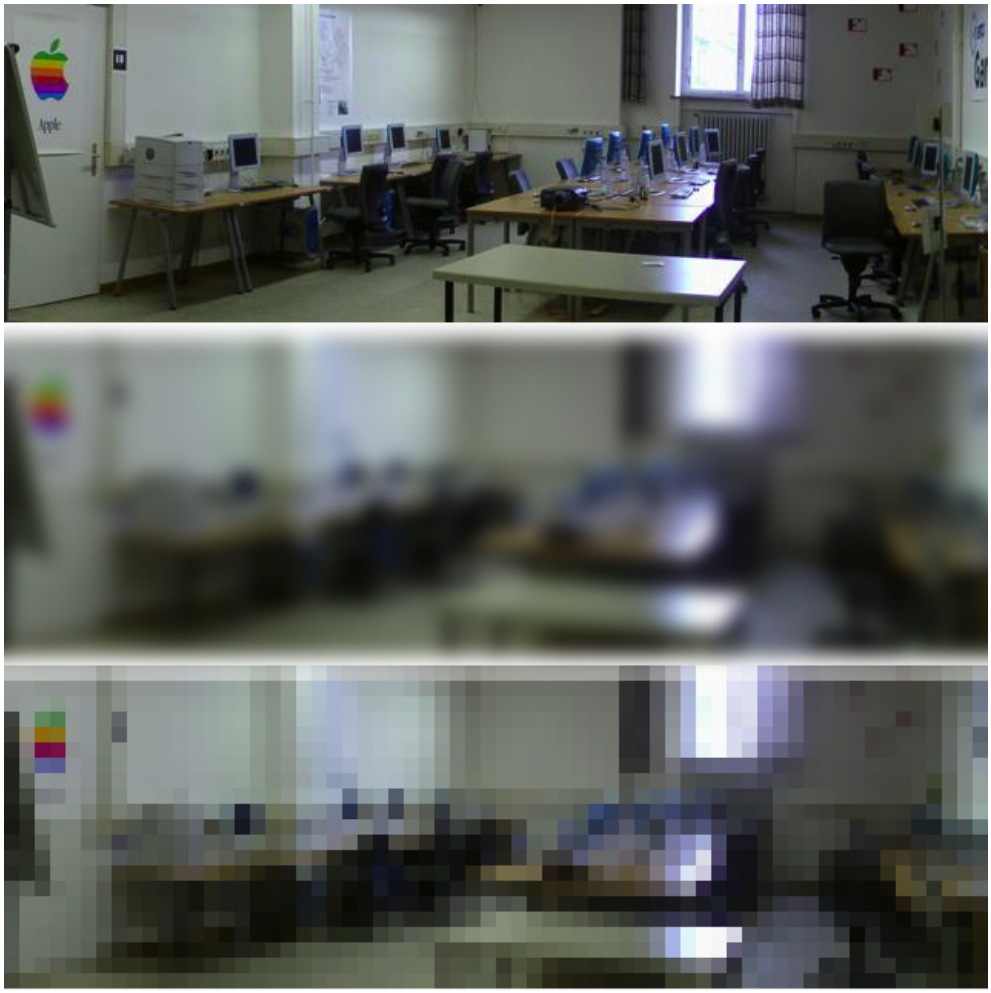

Figure 2.15: Applying a blur (middle) or a pixelize filter (bottom) to a video can help to preserve privacy

The study resulted in threshold levels for filtering that would balance privacy and awareness. Nevertheless images filtered to these levels (similar levels have been used in the image in Figure 2.15) do not reveal the identity of the actors nor do they provide enough information to identify the real-world objects being worked on.

Zhao and Stasko [85] have studied the effects of other filters, and found out that all filters would allow high activity recognition levels (more than 90%). Identity was much more difficult to recognize, but improved with familiarity. This result could be interpreted such as in environments where people are familiar with each other, video can be filtered to levels that preserve most of the privacy but still allows for some limited recognition of identity. The

Zhao and Stasko results also indicate that people are willing to reveal their identity while preferring to not disclose all of their actions.

The low recognition levels for identity and other awareness cues that require more detail raise the question about the general usefulness of literal awareness systems in settings where privacy considerations are important — such as activity in private offices rather than in public areas. On one hand one could substitute the video images with symbolic presence indicators. On the other hand one could allow the user to control the level of filtering and provide some symbolic presence indication. Systems that allow such level of control and combine video with presence indication based on video processing have been proposed in the past, e.g. by Lee, Schlueter and Girgensohn [48].

Video-based literal awareness systems are usually complemented by systems which transmit audio. The use of audio raises many privacy issues on its own. Rather than feeling observed, people feel that they are being eavesdropped. However, the filtering idea described above can be applied to audio signals in a similar way. One example of such an idea is an audio muffler proposed by Smith and Hudson [74].

Their system transforms speech into incomprehensible noises while preserving the original charaterstics such as the volume profile and the frequency distribution, from which the identity of the speaker can often be derived. They achieve their goal by preprocessing one short (6–8 seconds) sample of the user's speech from which the characteristic signal and the characteristic energy distribution is derived.

When the user is speaking, the system computes the current energy distribution and transmits it to the listeners, without transmitting the actual speech. On the receiving side, the preprocessed characteristic signal and energy distribution is retrieved and combined with the received current energy distribution to produce the output audio signal.

Unfortunately, the system relies on the ability of the sending side to identify the speaker reliably. Other than that, it seems to provide a solution to the privacy problems of audio-based awareness systems. The authors claim that their system is significantly less disruptive than systems that use unfiltered audio.

Revealing the identity and location seems to be controversial even with highly symbolic systems, such as presence and location revealing awareness systems based on active badges. Research by Harper [36] indicates that the reasons why people accept such systems have more to do with the organisational setting in which those people work rather than with the technology itself. Harper concluded that in an organisational setting with a high level of competition and low level of trust, a lot of people might just refuse to use an awareness system, whether it is literal or symbolic.

Thus, rather than making the awareness information just more symbolic and abstract, one also has to carefully choose which kind of information an awareness system distributes. One common approach that might help with acceptance is to distribute only information which already is public (privacy advocates will not agree fully with this statement).

The XWHO system follows this approach: it displays the login information for a set of computers in a laboratory. If the computers are configured in a way that allows each user to log into an arbitrary machine, than it is fairly easy for anyone to find out who is currently logged into a given machine anyway. XWHO just simplifies the process and provides an automatic overview of the logins.

Another approach to deal with the monitoring problem is to build the awareness systems in a symmetric way: one can only receive awareness information if one is willing to reveal awareness information about oneself. The highly popular instant messaging systems work that way: only after logging into the instant messaging server — and thus after revealing one's own presence — can one receive presence awareness information about others.

The above discussion leads to three principles of ensuring privacy:

1. symbolic (rather than literal) presentation,
2. publicity
3. and reciprocity.

Publicity means that the awareness system only uses information that already is public, and reciprocity means that only individuals willing to reveal information about themselves are able to use the system.

We assume that following these principles, we can address the privacy concerns users of awareness systems might have. This assumption is consistent with proposals found in the CSCW literature (e.g. [56]), and is sufficient for the technical discussion of awareness systems in this dissertation.

2.4.2 Information Overload

Information overload is another inherent problem of awareness systems. Awareness systems produce and forward large amounts of additional information to the individual and thus increase the amount of information an individual has to cope with. A situation where the individual is overwhelmed with information and information processing becomes erratic is described as Information Overload.

Information Overload is a rather recent phenomenon. However, it is not something that emerged as a result of the increasing use of the Internet in the recent years.

The number of scientists has doubled three times in the last 150 years — and the number of scientific publications has increased expotentially. While a 19th century physician could stay on top of all newly published research results relevant for his profession, today it would be impossible for him to read even a small fraction of all results published [30].

The number of books grew significantly in the last century, and continues to grow: the number of books published between 1950 and 1975 alone exceeds the number of books published before this period [68].

Today, large amounts of information are no longer published on paper but rather electronically. In January 2003, the search engine GOOGLE [3] allowed to search more than 3 billion web pages.

Users have to deal not only with information published on the Web, but also with other types of electronic information services, including Usenet newsgroups, Web-based forums and electronic mail. Users blame a low signal-to-noise ratio in electronic information media, where signal is meant to be the relevant information and noise is the irrelevant information contained in a given media (this usage is a generalization of the analog signal processing term).

In particular, studies show that users are dissatisified with the volume and contents of the information they receive. For example, users interviewed by Huang et al. [38] felt that a large part of the email they receive is irrelevant and distracts them from their real work.

This study probably does not even take into account the large volumes of unsolicited commercial email that increasingly fill the mailboxes of users around the world (some organizations have found that this so called "Spam" accounts for more than one third of all inbound email [76]).

Computer-based information processing methods could help increase the signal-to-noise ratio, if methods for intelligent filtering could be implemented. Still, one has to be careful not to put too much hope into new software solutions (including awareness systems): until now, computers seem to be more helpful in producing new content, while somewhat failing when expected to reduce the volume of information.

Thus, special care has to be applied to the design of awareness systems in order to avoid exacerbation of the information overload problem. Rather than simply gathering and forwarding awareness information to all users, an awareness system needs to provide individual filtering. An ideal awareness system only forwards those parts of available awareness information to a user that is relevant to that user in the current working context.

2.5 Summary

In this chapter, we looked at existing awareness systems and have presented a simple, yet comprehensive taxonomy for such systems consisting of four classes of awareness systems. The systems presented in this chapter all fit into our taxonomy.

Nowadays, literal awareness system using video can be easily built. Cheap webcams are ubiquitous, and some even include an embedded system for compressing and transmitting the digitized video directly via an integrated ethernet controller and TCP/IP stack (e.g. products from Axis Communications Corp.). The bandwith typically available today also surpasses the 56 kbps available for MEDIA SPACES by several orders of magnitude.

We have discussed the issues such as privacy and information overload that can arise when awareness system are deployed, because the issues currently faced by awareness systems are not technological: it is just as easy to build a literal system as a symbolic one. However, we have seen that literal systems in particular raise privacy and usability issues that need to be addressed before awareness systems become useful.

In this dissertation we address the privacy problem by favoring a symbolic rather than a literal approach to awareness and by following the reciprocity principle. Later, when connecting awareness event sources, we need to make sure that only information that is already public is forwarded to our awareness system, such that the publicity principle is respected as well.

To limit information overload, we will discuss in detail relevance assessment that allows to assign a relevance weight to each piece of awareness information that reaches the system.

Chapter 3

Experimental Context

In this chapter, we discuss the context for our experiments, using a series of distributed software engineering project courses as an example. In Section 3.1 we provide an overview and discuss the levels of distribution involved. Then we describe the educational goals (Section 3.2), the application domain (Section 3.3), the organizational structure (Section 3.4) and the methodology (Section 3.5).

We conclude this chapter with a description of the development (Section 3.6) and communication (Section 3.7) infrastructure used. This infrastructure forms the technical context for our prototype system (AWARENESS BUILDER, see Chapter 5).

3.1 Overview

The chair for Applied Software Engieering offers a four month software engineering laboratory project course, which exposes the students to the fundamental issues in software engineering.

These courses serve as the background to which the ideas from the previous chapter have been applied as well as the experimental context for evaluation of the prototypical solution, the AWARENESS BUILDER tool, which is described in Chapter 5.

The courses are distributed in both time and space. Some of them, e.g. the STARS course series, were globally distributed as they were offered in parallel on two continents, by two different universities: Technische Universität München, Germany (TUM), and Carnegie Mellon University in Pittsburgh, PA, U.S.A. (CMU). Some of the developers (students) worked at TUM, while others worked on parts of the systems at CMU.

In such global software engineering courses, the distribution in space oc-

curs on a large scale, spanning continents. Distribution in time was inevitable because the project teams were located in different time zones, that were six hours apart.

In addition to this kind of *large scale* space/time distribution, the distribution in time and space could be observed on a smaller scale, as well. Some project participants would only occasionally work in a colocated manner. Although the university provided meeting rooms and computer laboratories, many students preferred to work from home or meet elsewhere, introducing distribution in space on a smaller scale (this effect could most notably be observed at TUM, where there was no campus and/or no dormitories on campus).

The software enginnering project course is just one of many courses a student would take in a semester. Thus, no student could be expected to work full time on the project. Even if the students decided to meet and work together in a single place, their different course schedules would not allow enough common time slots to meet in person. Consequently, distribution in time was in effect even in the same time zone.

The Anytime-Anyplace-Matrix [54] (Figure 3.1) illustrates the general design space for distributed work support. In that matrix, the described software engineering laboratory courses fit into the upper right corner (different time and different place), due to the distribution in time and space described above.

At first glance, this makes our problem domain particular. Some practicioners dismiss the idea of distributing a project team altogether: for them, the attempt to have a project of e.g. 20 people, where the project management is in one building and the project members are dispersed across three other buildings is a recipe for disaster and the project is doomed to die the "distance death" (e.g. Wischnewski uses the German term "Entfernungstod" in [82], page 40).

However, there are a number of good reasons why a project has to be distributed. Mergers and acquisitions, global sourcing (i.e. "off-shore" development) and attempts to gain flexibility through virtual organization often result in project teams distributed in space and time (often across multiple time zones) [62].

Still, colocation helps projects to succeed as it allows all project participants to communicate with one another freely and easily, without having to resort to mediated communication. It also allows project managers to be present where all the action is – Management By Walking Around [60] is easy if it just means going down the floor. Proponents of Extreme Programming [10] claim significant productivity gains in software development with techniques that require physical colocation.

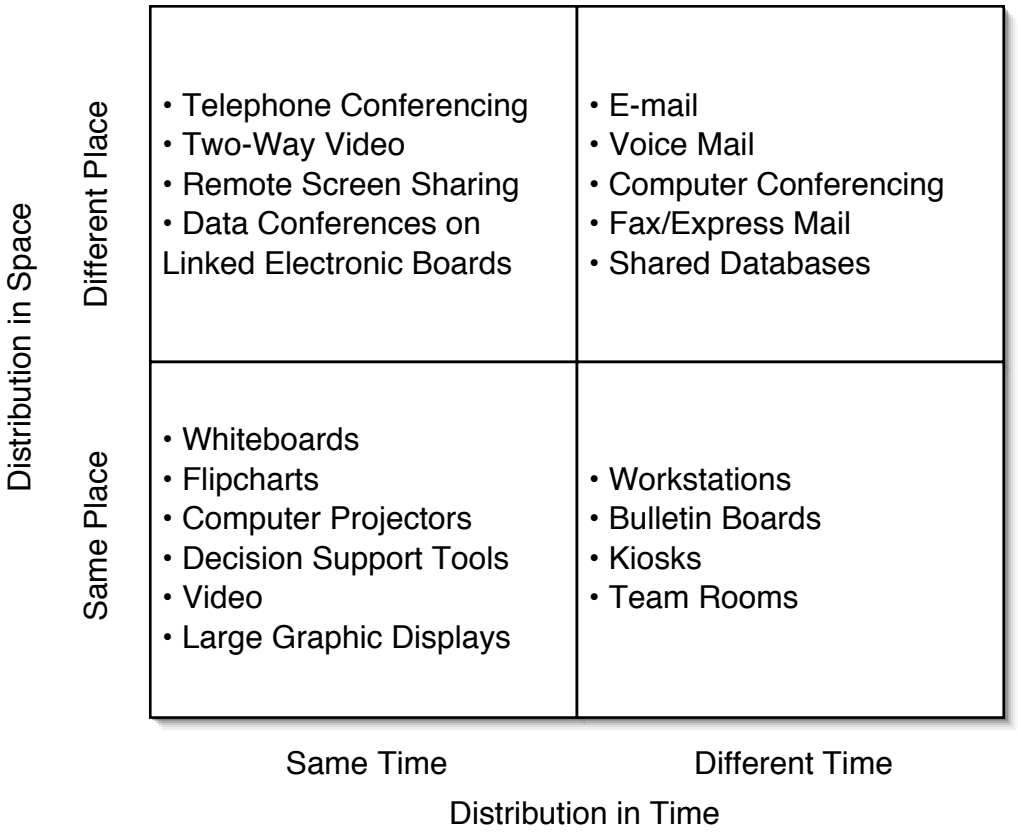

Figure 3.1: The Anytime-Anyplace-Matrix (adapted from [54])

Unfortunately, as we have seen, to some degree, distribution is inevitable. Only small teams can be colocated on one floor, if they can get adjacent offices. Larger teams need to be dispersed across floors or even buildings, as soon as they reach a size, that will not fit on a single floor. Often several small teams start by sharing a floor in an organization, but when one of them grows, the new members of the growing team have to be located at some remote place, because removing the other teams may not be an option.

Studies have shown that communication frequency tends to drop sharply with increasing distance [45] – researchers who had offices next to each other had approximately twice as much communication as those whose offices were on the same floor, but at a greater distance. A distance greater than 30 meters between offices or offices on different floors seems to be almost as bad for communication as offices in different buildings, with communication reduced by up to 80%. This phenomenon was found to be valid not only for face-to-face communication, but also for phone calls as well as electronic

mail messages.

Thus, we believe that there are more projects that are similar to our software engineering courses in terms of distribution that one may initially assume.

Also, while being conducted in a university environment, the courses provide the students with a realistic experience of a large scale software develepment project. Thus, some observations and assumptions used with this problem domain may be generalized for industrial software development projects conducted by organizations outside the university environment. This is in particular true for industrial real-world projects that develop prototypes rather than products.

The evaluation of the tool built to fit this problem domain also took place in the context of university software engineering projects, rather than in the context of industrial software engineering projects, of course. Still, the courses provided an environment that is sufficiently realistic and controlled. While it is not completely representative for all industry projects, we tried to sort out the course related artifacts from the general observations during the interpretation of the results (see Chapter 6).

3.2 Educational Goals

The goal of the courses is to teach skills needed by software engineers through example and practice. Before the students take the course, they are expected to already know how to program in at least one programming language and to either already know the language used during the course, or being able to learn it rather quickly during the early phases of the projects.

The students need to be in their third or fourth year at the university to take the course, so at least some of them have already had the chance to take advanced classes, such as classes on databases or networking. Basic textbook-level knowledge about software engineering is taught in parallel.

The students embark on a real project, involving a real-world problem. On such a project, the students can apply their software engineering knowledge gained from books and lectures and experience what works and what does not. The experience of living through a project allows them to apply software engineering concepts and methods in practice.

The phenomenon that students are not expected to know first hand is the communication overhead of larger projects. They are used to work on toy problems and assignments that are crafted in such a way that they can be completed by one person in a reasonable time frame. While they might have read or heard about communication problems of larger projects, most

of them cannot picture the consequences.

3.3 Application Domain

The software engineering courses usually rely on a external client to provide a real-world problem. Depending on clients' needs, the application is different each time a software engineering course is offered. As examples, we take a closer look at the application domains of three courses (in chronological order): PAID, STARS, and TRAMP.

The PAID course series started in Fall 1998 with a software engineering course at the Carnegie Mellon University in Pittsburgh, PA, U.S.A. (CMU course number 15-413). PAID stands for Platform for Active Information Dissemination. The goal of the project was to develop a prototype of a novel means of disseminating maintenance-related documentation for the Daimler-Benz (now DaimlerChrysler) corporation.

This documentation consisted of service, parts and vehicle documenation. Depending on the type of information the company utilized a variety of different distribution channels for aftersales information. When the project started, the manufacturer used a set of CD-ROM's, that was assembled and sent to the dealerships on a monthly basis. This distribution method was considered very slow, inefficient and expensive by the client.

Over time, aftersales documentation and other types of new documentation needed to be included in the dissemination process as well. The amount of aftersales information was increasing due to the introduction of new aftersales information systems and due to the steady introduction of new models.

The client stated a list of requirements, including:

- minimize the proliferation of different distribution channels for aftersales information,
- facilitate the distribution of aftersales information and applications to business units located at all enterprise levels (e.g. headquarters, wholesale, dealers),
- establish a powerful and secure standard for information/software distribution on the Daimler-Benz Intranet and its extension the Daimler-Benz Extranet,
- customize and adapt the distribution of aftersales information according to the specific business requirements of the accessing applications respectively users,

- minimize the costs for the implementation of new aftersales applications and associated data distribution infrastructure facilities,

- minimize the administration costs at the dealer as well at the headquarters.

During the first project of the PAID course series, the students completed a thorough requirements analysis that resulted in a basic system design and a first, prototypical implementation of the system.

This first phase (or iteration) of the PAID project was overlapped by a second phase at the Technische Universität München, Germany (Softwaretechnik Praktikum) in winter semester 1998/1999. Due to the overlap, the TUM students were able to build on the work of the CMU students, refine the architecture of the system and improve the implementation prototype.

The third iteration of the PAID sequence was an advanced software engineering course at the Carnegie Mellon University in Spring 1999 (CMU course number 15-499), where data mining and machine learning algorithms used in the system were improved.

The fourth iteration of the PAID project was a field test at selected Daimler-Benz dealerships in Germany and USA in August 1999. The field test was done as diploma theses by selected students from the courses.

The STARS project started in Fall 1999 with an advanced software engineering course at the Carnegie Mellon University (CMU course number 15-499). STARS was an acronym for Sticky Technology for Augmented Reality System. The goal of the STARS project was to develop a prototype of a visionary system supporting F/A-18 "Hornet" maintenance technicians for the U.S. Navy using advanced wearable computing and augmented reality technology.

STARS attempted to reduce the cost of logistics support and maintenance while improving or at least maintaining current aircraft readiness. To address this problem, the improvement of two major processes was proposed: developing and managing interactive electronic technical manuals (IETMs) and performing maintenance with advanced capability IETMs.

The augmented reality technology component involved advanced real time video and sensor data processing that allowed to track the user's movements and display context information such as work order steps from an IETM directly into his field of vision on a head-mounted display.

Again, during the first iteration of the STARS project, the students completed the requirements analysis and designed and implemented a prototype of the system envisioned by the client. Their work was continued in a second iteration by the participants of a software engineering course at the Tech-

nische Universität München (Softwaretechnik Praktikum) in winter semester 1999/2000.

During the second iteration the problem domain was broadened to include the maintenance of other complex systems, nuclear power plants in particular — with Siemens KWU being the customer for the German part of the project.

The STARS project concluded with two parallel iterations at CMU and TUM in the winter semester of 2000/2001 where the architecture was refined and new features such as multimodal user input were implemented.

The TRAMP project took place in the winter semester 2001/2002 at the Technische Universität München. Similar to the scope of STARS, the goal of the TRAMP project was the development of a system enabling the remote repair and maintenance of complex systems. TRAMP stood for Traveling Repair and Maintenance Platform.

The envisioned user of the TRAMP system is the field technician who needs to access blueprints and engineering information, to find and communicate with remote experts via audio and video streams, and to manage context associated with several repair and maintenance tasks. Also, the technician must be able to accomplish these tasks at the location of repair using networked devices that support wireless networking such as WLAN, GSM/HSCSD/GPRS and, eventually, UMTS.

The specific goal of the course was to develop a functional prototype based on UMTS to realize a mobile garage. Mobile mechanics can service, diagnose, and repair a vehicle that has broken down, regardless of it's location. The results of the praktikum was a prototype system and a generic architecture for complex mobile systems.

The client for the TRAMP project was Inmedius, a company that develops mobile systems for the maintenance of complex systems, such as aircraft, powerplants, or ships.

3.4 Organizational Structure

Traditionally, division of the labour into well-defined subtasks, assignable to individual developers [61] is used to seize the opportunities of collaboration and to prevent the problems related to communication overhead. For complex systems, the division of labor needs to be done recursively on several levels of abstraction.

In general, division of labor can occur using quantity or type criteria. While quantity based division of certain tasks (e.g. testing in software engineering) is quite fruitful (e.g. Open Source software), it does not allow to fully realize the benefits of specialization. Unfortunately, poorly struc-

tured and changing tasks require broad qualifications and adversely effect the ability to specialize.

Specification and design of a software system seem to be such poorly structured and changing tasks. Brooks states [14] that the actual coding of a computer program is easy compared to the inherent difficulties that lie in the area of designing it, that is listing the design goals and describing the system architecture in terms of the subsystem decomposition of the system considered. He argues that we therefore cannot hope for speeding up the development process as a whole by applying any single method. Especially the design of a software system cannot be easily divided into subtasks and therefore defies the benefits of specialization.

In fact, the design process for a software system is very similar to the process of dividing labor itself, respectively designing an organizational structure. The Transaction Costs theory [61] suggests that an ideal subdivision of tasks within an organization or between organizations comes along with as few simple transactions as possible. This property can be achieved by either minimizing the interdependencies between tasks or by dividing the tasks according to knowledge-maturity.

Minimizing the interdependencies between the subtasks, while maximizing the interdependencies within a subtask is similar to the design concept of minimizing the coupling between subsystems and maximizing the coherence within a subsystem. The ideal subsystem decomposition has subsystems that are loosely coupled and have high coherence [16].

The coupling metric measures the intensity of dependencies between two subsystems. Loosely coupled subsystems have only small dependencies between them and changes in one subsystem have little impact on the other one, whereas in strongly coupled subsystems a change in one subsystem usually entails a change in the other one.

Coherence measures the intensity of dependencies within a subsystem. Highly coherent subsystems contain only classes that perform similar operations, work with the same data or are in some other way closely related to each other. Subsystems containing unrelated classes that appear to be coincidentally grouped together have low coherence and are hard to maintain. Studies have shown that routines with low coherence tend to have more defects than highly coherent routines [19].

Thus, the concepts of coupling and coherence are directly related to the suggestions the Transaction Costs makes on the subdivision of tasks in human organizations.

The concept of knowledge-maturity (german "wissensökonomischer Reifegrad"[23]) is another organisational theory idea that translates well into the software engineering domain. It is based on the observation that some

parts of human knowledge cannot be easily articulated or described. This so called implicit or tacit knowledge needs to be expensively learned in a master/apprentice relationship that introduces high transaction costs in case the knowledge needs to be transferred from one person or team to another. It is therefore desirable to minimize the necessity to transfer implicit knowledge when dividing up labor.

Implicit knowledge can be transferred more efficiently if it is encapsulated in work products that can be used and processed without this knowledge. Work products which cannot be used without intimate knowledge about their production have insufficient knowledge-maturity. Work products which can be used easily without such knowledge have reached a sufficient knowledge-maturity. A work product can reach knowledge-maturity in stages and all these stages are potential task decomposition points.

The notion of knowledge-maturity can be translated into terms familiar to software engineers as Parnas' information hiding principle [57] . The idea behind information hiding is to encapsulate areas that are likely to change, have complex data or logic. Users of a subsystem should not have to worry about such issues but rather have a high level interface to that subsystem, which hides those issues from them. Information hiding is known to improve the quality of software [12] and is a key principle for object-oriented design and programming.

Software architecture is expected to incorporate the concepts of minimum coupling, maximum coherence and information hiding. Such a software architecture defines a *natural* division of labor along the boundaries of the subsystems. This natural organization structure appears to be optimal in terms of the transaction costs due to the dualisms described earlier.

The projects are therefore organized into several teams, five to nine members each. The teams fall into one of two categories: subsystem teams and crossfunctional teams.

The subsystem teams are teams that result from the natural organizational structure decribed above. The preliminary architecture that this structure is based on is always designed by the project leader of the given course.

In addition, a number of crossfunctional teams is formed. The membership in a crossfunctional team is an additional role for members of the subsystem teams. The crossfunctional teams focus on issues such as architecture or testing, that span the boundaries of the subsystem teams.

The resulting organizational structure is a matrix organization with subsystem teams on the one dimension and crossfunctional teams on the other dimension. The team lead role is filled by one of the student members of the team and often rotated during the course.

Each team is supported by a coach who provides guidiance and feedback

for the team. The coaches are usually students who have participated in a software engineering project course before. The additional project management training needed is provided in a separate mandatory seminar on project management (Hauptseminar Project Management: Advanced Topics) that runs in parallel to the course.

3.5 Methodology

The courses follow the object-oriented methodology described by Brügge and Dutoit in [16]. The development process is split into a requirements elicitation and analysis activity, a system design activity, an object design activity and an implementation activity. In addition managerial activities such as rationale management, testing and software configuration management are performed.

The life cycle of the project course consists of a series of cycles where each cycle includes the sequence of the development activities that result in a prototype that is delivered to the client. Each subsequent cycle incorporates the feedback received from the client, until the project ends with the Client Acceptence Test, where the final prototype is presented to the client. E.g. in TRAMP, there have been four cycles named by the students with the prototype names "straw", "wood", "brick", and "steel" as figurative descriptions of the intended levels of completeness.

During requirements elicitation and analysis, the students describe the functionality of the system from the user's point of view. Based on a scenario, the boundaries of the system are defined. Scenarios are then abstracted into use cases and the relationships between use cases, such as include and extend are analyzed.

After the use case model is reviewed, participating objects and associations between them are identified and an initial object model is defined. This initial object model helps to find a common terminology for both the developers and the client and serves as a first step towards a complete analysis model.

The initial object model is refined by adding additional associations and attributes to the objects. At the same time, the objects are structured into entity, boundary and control objects to encapsulate areas that are likely to change.

These models are documented in a Requirements Analysis Document (RAD) which also includes nonfunctional (e.g. expected performance) and pseudo requirements (e.g. programming language).

During the system design the students derive a set of design goals from

the nonfunctional requirements. Then they describe an initial subsystem decomposition and map the subsystems to processors and components. They also make decisions about persistent storage, define access control policies and control flow strategies. Usually new use cases come up during this phase — e.g. start and shutdown of certain parts of the system — and are formulated as boundary conditions. The results of the system design activity are documented in the System Design Document (SDD).

During object design, the students bridge the gap between the system envisioned during the requirements analysis and the hardware/software platform defined during system design. Make of buy decisions are made about off-the-shelf components such as class libraries and application frameworks. Also, depending on the project specific set of design goals, the object model is optimized for properties such as extensibility, understability and/or performance. The results of the object design activity are documented in the Object Design Document (ODD). The ODD is seldom a separate document like the RAD or the SDD. Rather it is often automatically generated by a tool as an extension of the RAD.

During implementation, the students map the object design into source code. This activity includes creating a set of source files, compiling them into a runable system and testing them. The result is a set of source files and a set of built binaries that form the executable system.

3.6 Development Infrastructure

The project participants are being provided with meeting facilities, computing labs, and development tools for the duration of the project.

The set of editors, compilers and build tools in the lab depends on the operating system used and the programming language chosen. Usually, the students are free to choose any editor they like (e.g. vi, emacs etc.), but are required to use a certain compiler (e.g. gcc) and a certain build system (e.g. make) to build the deployment version of their system.

Multi-site projects involving project teams at TUM and CMU can not rely on a homegenous infrastructure provided by the computing lab, because the resources available at each site are different.

Thus, the editors, compilers and build tools used during the projects were quite heterogenous and changed not only from project to project but also during a project.

One development tool has been used extensively in PAID, STARS and TRAMP. This tool was the software configuration management (SCM) system CVS. CVS or Concurrent Versions System is a popular open source SCM

tool that is available in a client/server versions on most Unix variants. CVS client functionality is available on almost any platform (Figure 3.2 shows a Web-based client).

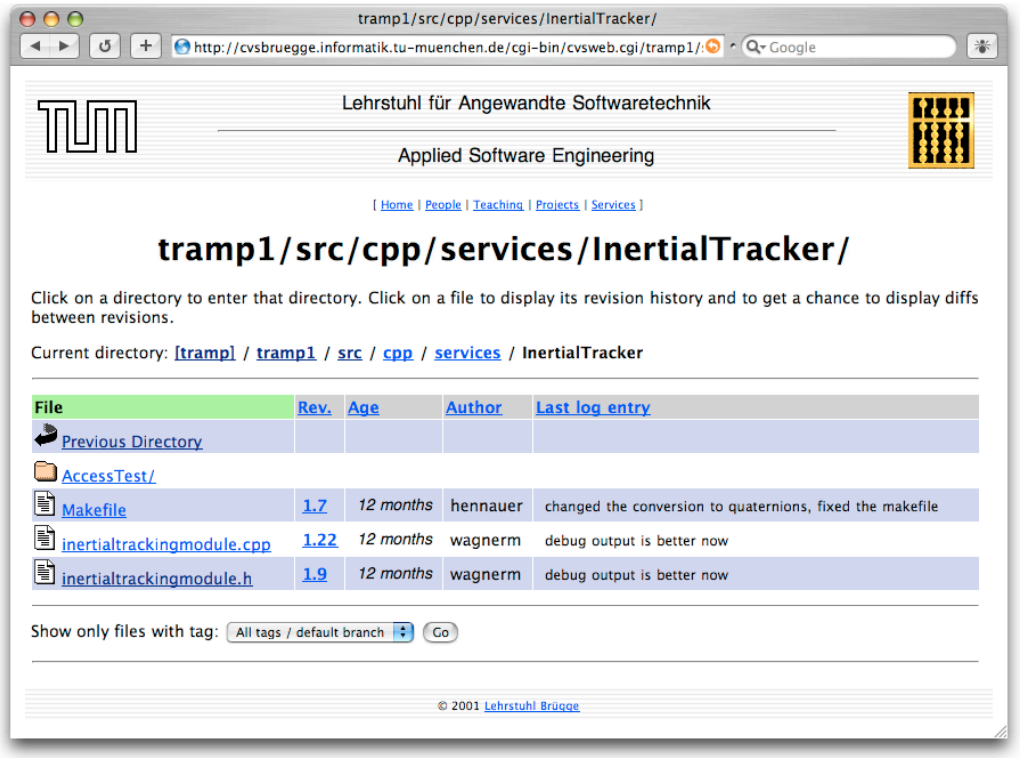

Figure 3.2: CVSWEB provides a Web-based interface to CVS repositories.

The focus of CVS is on record keeping and collaboration. Contrary to other SCM tools that use a pessimistic change control policies involving locking a file, modifying a file and than unlocking a file, CVS uses an optimistic approach, allowing concurrent modifications, and merging of changes in case of conflicts. Although this policy can not prevent conflicts, many developers prefer this approach: with other systems, developers often lock a file and then forget to release it, causing disruption in the project schedule. With CVS such situations are minimized.

A typical use case for CVS consists of the following steps:

- Developer requests (checks out) a working copy. The working copy is the developer's workplace library where she makes her local changes.
- Developer makes changes to her copy

- Developer commits (or checks in) her change along with a log message. The master copy is updated automatically. Technically speaking an new revision is added to the master repository and assigned an internal version identifier.

- Meanwhile, other developers can have CVS update their working copies to incorporate the recent changes.

3.7 Communication Infrastructure

During the project course students have access to common communication tools such as phones, email and video conferencing. In addition the students are provided with a set of web-based discussion forums, built on top of Lotus Notes Domino technology (Figure 3.3).

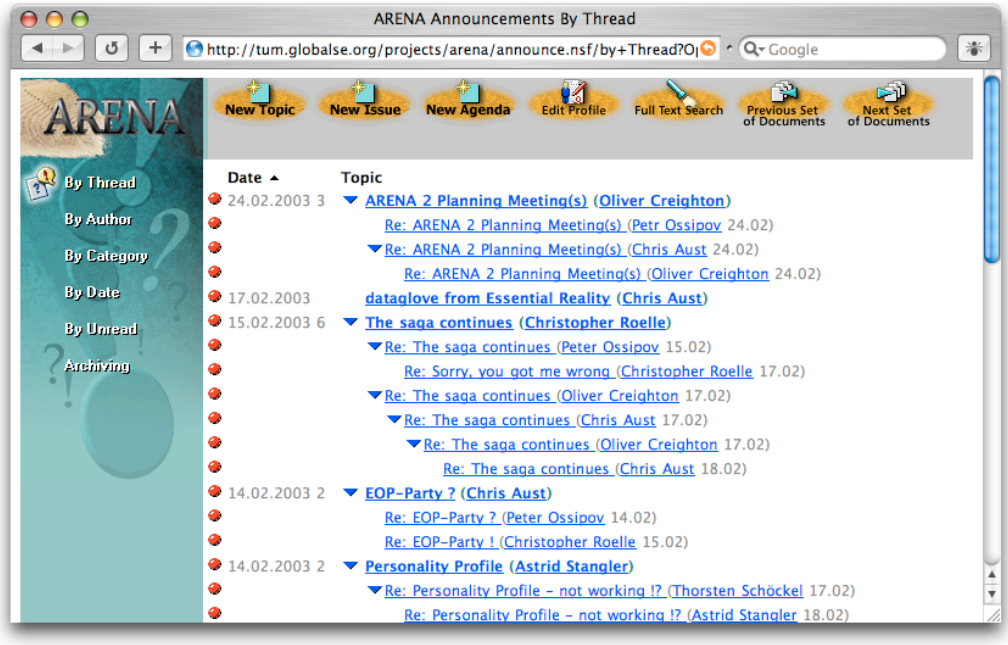

Figure 3.3: A web-based discussion forum from the ARENA project

The structure of the discussion forums follows the organizational structure of the project. Thus, each team has a separate discussion forum, where topics relevant to the given subsystem can be discussed. This set of discussion forums is augmented by several project wide forums and a couple of forums with limited access.

The project wide forums usually include an announcement board, where important announcements can be posted by the project management. A client forum exists for discussions with the client and another forum is always dedicated to general discussions about the project.

3.8 Summary

In this chapter, we have shown that even those software development projects that do not employ large scale distribution (such as spanning continents and time zones) can suffer from small scale distribution (such as spanning floors and buildings. We believe that awareness systems such as those discussed in the previous chapter may help to alleviate the impact of physical distribution on all kinds of projects.

The project courses described in this chapter use a large and changing number of development and communication tools, but two major tools — CVS and LOTUS NOTES — have been used again and again as the cornerstones of each project conducted between 1998 and 2003. These two tools are therefore the natural candidates for adding awareness features to them.

Next, we will present the requirements for Awareness Systems that we derive from our application domain discussed in this chapter and the group awareness basics presented in the previous chapter.

Chapter 4

Requirements for a Symbolic Awareness System

In Chapter 2, we have seen that awareness systems can be classified into one of the four classes of an awareness presentation matrix. In Chapter 3 we discussed software engineering project courses that serve as the application domain of our awareness theory. In this chapter, we discuss the requirements for symbolic awareness systems for distributed software development projects.

After stating the non-functional requirements (Section 4.1) that we derive from the material from Chapter 2 and Chapter 3, we present a simple use case model (Section 4.2). Then we discuss three key functional areas that we have identified as important for all symbolic awareness systems: awareness data collection, relevance assessment and awareness information dissemination (Sections 4.3-4.5).

4.1 Non-Functional Requirements

From the group awareness basics presented in Chapter 2 and our problem domain presented in Chapter 3 we derive the follwing non-functional requirements.

User interface device independence: The architecture should not be bound to any specific type of presentation devices, but allow both traditional GUIs as well as novel ambient devices as displays, in order to support the full spectrum of awareness systems.

Disconnected display support: To allow the widest possible selection of presentation devices, we should not assume that these devices will be always available online. Rather, displays may or may not be available all the time and might go randomly offline and then online again.

Privacy control: The privacy principles symbolic presentation, publicity and reciprocity have to be considered. To ensure this priciples, sufficient access control mechanisms need to be put in place.

Information overload control: Filtering of awareness information must be possible, without requiring too much manual configuration on the users part and sacrificing usability. To allow experimentation with different filtering methods, the architecture has to allow easy modification of these methods.

Seemless integration: The awareness system should fit into an existing development and communication environment without replacing it or interfering with the regular way of doing things (e.g. requiring the user to take extra manual steps to create `AwarenessEvent` objects). Also, the impact of an awareness systems on existing applications should be kept to a minimum and there should be no sensible performance hit on the applications already in use (i.e. users should not notice a difference between operation with or without the awareness system in place).

Problem domain independence: Initially, we attempted to design our awareness architecture in a problem domain agnostic way. However, we discovered a number of issues that can be resolved in many arbitrary ways. Rather than making such an arbitrary choice in the design stage, we show that these issues need to be carefully addressed with the application domain in mind in the implementation stage. To deal with this requirement the architecture needs to be extensible.

4.2 Use Case Model of Awareness Systems

A computer-based awareness system tries introduce the feeling of physical proximity in a physically distributed setting. It needs to collect information through monitoring the actions of a number of users which we call subjects and deliver them to remote users which we call observers. Thus, it needs an information collection functionality and an information presentation, or more generally, a dissemination functionality.

The information collected on the subjects' actions consists of a stream of individual `AwarenessEvent` objects that form the input of the system. The information disseminated to the observers consists of a stream of new `AwarenessNotification` objects which form the output of the system.

`AwarenessNotification` objects are created in the awareness system to inform a particular observer about a particular `AwarenessEvent` object. We describe the `AwarenessNotification` class in more detail later in this chapter.

However, the inputs from the subjects are not simply forwarded to the observers. First, for each known observer, the system needs to assess the relevance of the particular `AwarenessEvent` object. Only if the `AwarenessEvent` appears to be relevant for that observer, the awareness system will create a `AwarenessNotification` object to inform the observer about the `AwarenessEvent` object[1] (Figure 4.1).

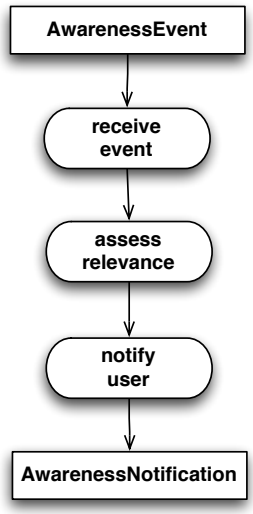

Figure 4.1: The inputs from the subjects are not simply forwarded to the observers. First, for each known observer, the system needs to assess the relevance of the particular `AwarenessEvent` object.

Due to this relevance assessment functionality, we could call the stream of input objects (mapped to `AwarenessEvent` objects) also *awareness data*. The term *data* implies that these inputs are syntactically correct, but not yet semantically tested information representations. Consequently, outputs

[1]This functionality somewhat mirrors the biological selective perception process

of the system could be called *awareness information* (mapped to `Awareness-Notification` objects) in the semiotical sense.

We conclude that a computer-based awareness system can be described in terms of three key use cases: the collection functionality, the relevance assessment functionality and the dissemination functionality (Figure 4.2).

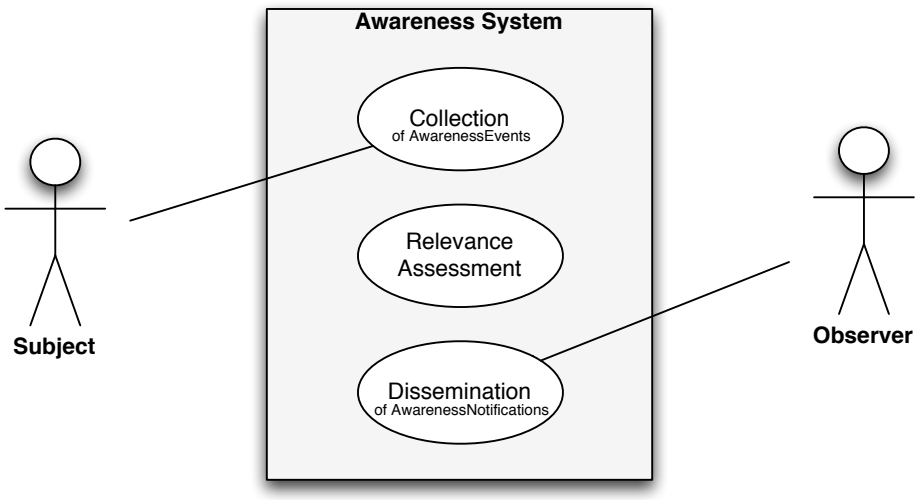

Figure 4.2: Each awareness system has two main actor types: the *Subject* (left) and the *Observer* (right). The main use cases are collection, relevance assessment and dissemination of `AwarenessEvent` objects (see also Figure 2.3).

The three use cases can me be mapped to three services, that constitute an Awareness System (Figure 4.3).

4.3 Collection of Awareness Data

4.3.1 Extended Workspace Model

The collection functionality of a computer-based awareness system has to provide generic services for monitoring the actions of subjects and capturing awareness data. It needs to provide facilities to capture these data from both ambient sensors (e.g. active badges systems) and from the applications used by the subjects.

In Chapter 2, we introduced a model for the `Workplace` which consists of `Person` and `Entity` objects. Typically, there is a number of distinct sub-

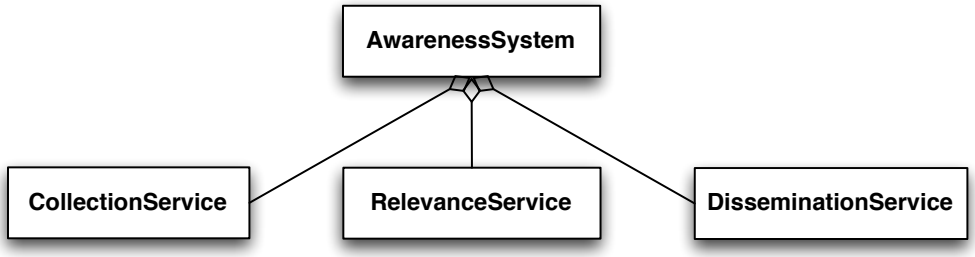

Figure 4.3: An `AwarenessSystem` consists of a `CollectionService`, a `RelevanceService` and a `DisseminationService`.

classes of `Entity`, each supporting a different set of methods. For example, a `RoomEntity` subclass of `Entity` might have the operations `enter()` and `leave()`, while might not support change().

In this model, each time a subject of type `Person` invokes an operation on an `Entity`, an event represented by a class `AwarenessEvent` is created. While we mentioned that a monitoring mechanism is required, we did not look in detail how this monitoring has to occur.

Earlier, we made a distinction between real-world entities and "virtual" entities that do not exist in the real-world. Users can not manipulate these "virtual" entities directly, but rather have to use applications that provide them with a set of operations. However, applications by themselves are not interesting in terms of awareness. Consider a file editing operation: what might be interesting the most for others is the fact that somebody has manipulated a file, while the type of editor that was used to perform the editing might not be interesting at all. Therefore, we have not included applications in our `Workspace` model.

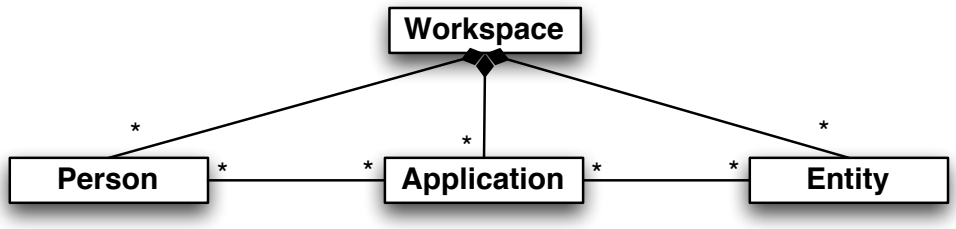

Figure 4.4: The simple Workspace model from Chapter 2 can be extended to include applications.

Still, applications can be included in this model as an intermediate class `Application` that acts as a facade to arbitrary groups of `Entity` objects (Figure 4.4). In this extended model, subjects do not invoke operations upon entities directly, but rather use operations of the application classes to manipulate the entities. Figure 4.5 shows an example where the class `FinderApplication` is a facade to `Entity` subclasses of type `Drive`, `Removable`, `Printer`, and `File`. Users cannot access the respective objects directly but rather have to use an instance of the `FinderApplication` class.

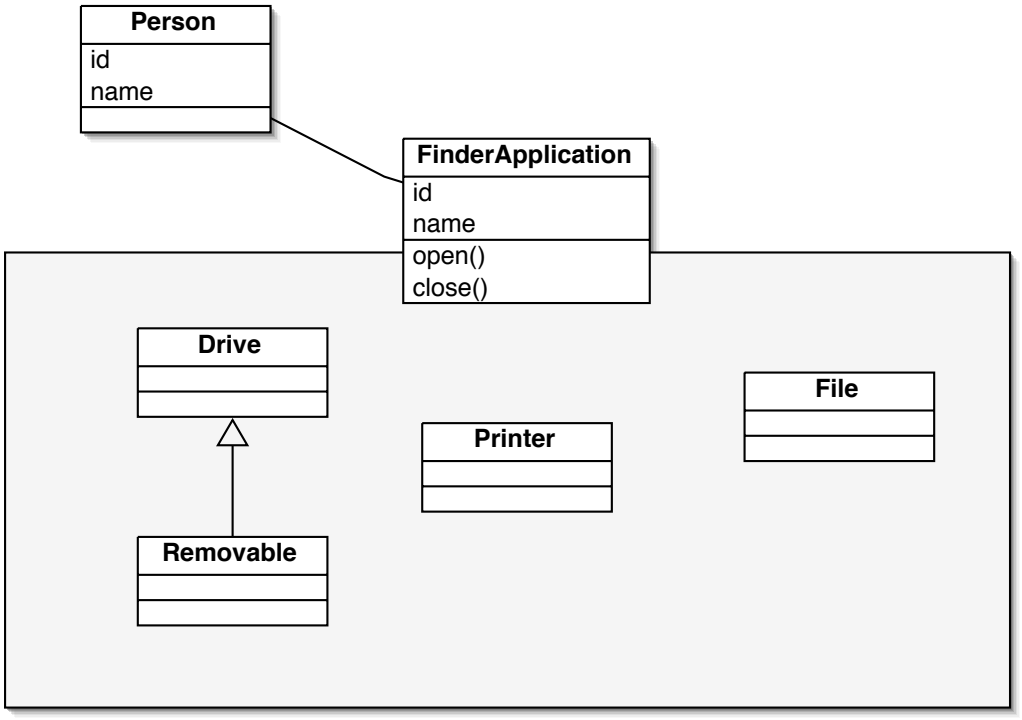

Figure 4.5: Applications can be viewed as a higher-level interface to a set of entity subclasses.

4.3.2 Awareness Sensors

With a `Workspace` consisting of people, applications and entities, activity monitoring can occur in three different ways: through user sensors, application sensors or entity sensors (Figure 4.6). User sensors are end user applications that allow the user to manually create events. Application sensors are extensions that are built into existing end user applications through their extension mechanisms. Finally, entity sensors monitor entity of a certain type

and create events upon detecting an operation invocation or a change in the `Entity` state.

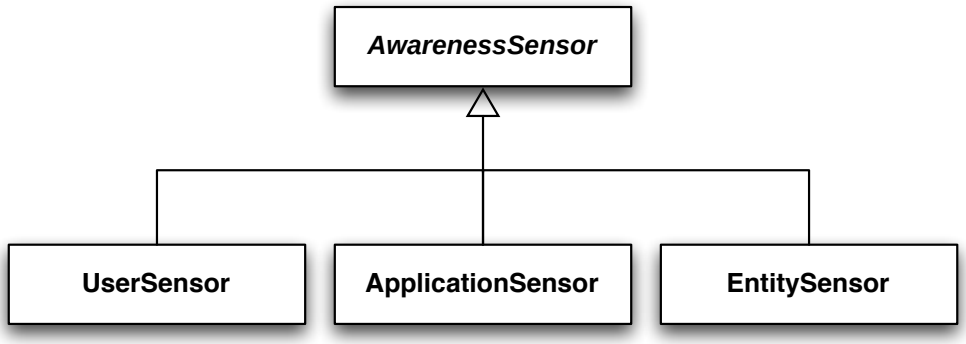

Figure 4.6: The Awareness Sensor as an abstract class

The first option — user sensors — would require that all users execute an additional awareness specific operation either before or after invoking an operation. Its advantage is that it makes neither assumptions about the applications nor the entities used. This option does not seem practical, because it would impose an additional workload on the (human) user and violates the *seamless integration* requirement.

The second option — application sensors —- requires that the application is modified so it creates an event every time a user uses it to manipulate an entity. This option is more practical, because it would allow to collect events without distracting the user nor imposing any additional workload and thus would honor the *seamless integration* requirement. Unfortunately, many applications lack extension mechanisms that would allow such modification.

The last option — entity sensors — requires the modification of the entities' behaviour. In practice, entities are often either passive (e.g. files), or internal to applications. In the former case, such entities have no behaviour on their own, and rely on the applications to manipulate them. Such entities can not be modified to create events but rather have to be monitored and checked for modifications constantly. In the latter case, internal application entities can not be accessed from the outside at all without modifying the applications.

In consequence, neither choice appears to be free of problems. In practice one needs to delay the decision about which approach to use to after the workspace has been defined in terms of concrete, domain specific applications and objects.

4.3.3 Collection Protocols

The collection functionality can be modeled as a set of awareness sensors, capable of observing a user's actions (operations on entities) and creating awareness events on these actions. Awareness data from these sensors is forwarded to a `CollectionService` object which forwards them to the relevance assessment service (Figure 4.7).

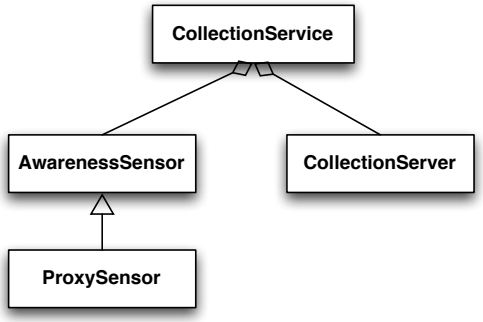

Figure 4.7: The Collection Service consists of a number of sensors and a collection server — proxy sensors are optional.

The communication between the sensor processes and the main awareness collection process can follow a simple client/server architecture. The `AwarenessSensor` instances are the clients and the main awareness collection server is the server. The server has to provide only one operation: `receiveEvent()` (Figure 4.8).

The platform on which the awareness sensors will have to run and the programming language in which the awareness sensors will have to be implemented is not known in advance.

Thus, the protocol the awareness sensors will have to use to communicate with the collection server has to be platform independent and language independent (this is another consequence from the *seamless integration* non-functional requirement).

There is a number of platform independent protocols that could be used for this communication (e.g. CORBA, SOAP, XML-RPC, Distributed Objects, Sun RPC, etc.). The selection of a specific protocol does not have an impact on the rest of architecture, as long as the criteria stated above are met.

Whichever protocol is chosen, there might be cases where we need to integrate an application that cannot support that protocol. In that case, a proxy sensor can be implemented that communicates with the awareness

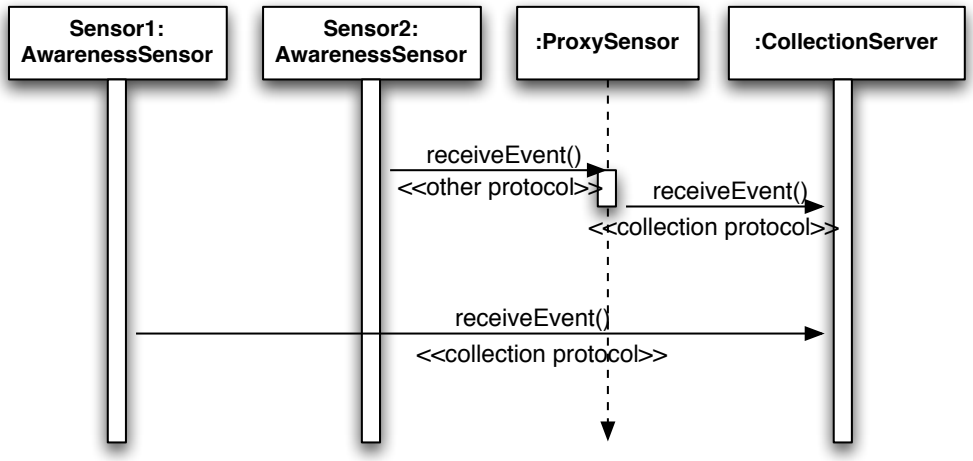

Figure 4.8: The communication between the sensor processes and the main awareness collection process can follow a simple client/server architecture.

system via the chosen standard event collection protocol and with the application in some other way. So, multiple protocols could be selected and supported easily.

4.4 Relevance Assessment

In this section we cover in detail the concepts that allow to assign individual relevance weights to awareness events. We use information retrieval concepts [30] and earlier work on relevance assessment in awareness systems (e.g. [18], [69]) as a starting point.

We begin with the explanation of basic information retrieval terms and discuss their implications on the relevance assessment (Section 4.4.1). Then we introduce the notion of interests and present formulae that can be used to effectively calculate a relevance function (Section 4.4.2). Next, we introduce concepts that allow to automate the process of defining individual interests (Section 4.4.3) and that allow to specify broad interests (Section 4.4.4 and Section 4.4.5). We conclude with a summary of the relevance assessment process.

4.4.1 Recall, Precision and Relevance

Relevance assessment has to make sure that the users get to see only the awareness events that are interesting and relevant for them.

This can viewed as an instance of a general information retrieval problem. In information retrieval, the most metrics for the quality of a query result set are the recall and precision achieved.

Recall relates the number of relevant data elements found to the total number of *all* relevant data elements. Precision relates the number of relevant data elements found to the total number of all *found* data elements. Recall is therefore sometimes referred to as the completeness and precision as the purity of a search result set.

More formally, recall and precision can be defined as follows [30]:

Definition 4.1 (Recall and Precision) *Given a query q and a set T of data elements, let $R \subseteq T$ be the set of elements relevant for q and $Q \subseteq T$ be the result set for q. Then the recall r is defined as $r = \frac{|R \cap Q|}{|R|}$ and the precision p is defined as $p = \frac{|R \cap Q|}{|Q|}$.*

There is usually a trade-off between recall and precision scores. In practice, result sets with high recall scores tend to have low precision scores, e.g. a result set containing all available data elements will have a recall score of 100%, while often having a low precision score: if $Q = T$ then $r = \frac{|R|}{|R|} = 1$, but $p = \frac{|R|}{|T|}$. If the number of relevant data elements is only a small fraction of the total number of available data elements ($R \neq T$ and $|R| \ll |T|$) then precision p will be small.

Conversely, result sets with high precision scores often have low recall scores, e.g. a result set containing just one relevant data element (such as only the first relevant element found) will have a precision score of 100%, but a low recall score: let $e \in R$ and let $Q = \{e\}$ then $p = \frac{1}{1}$, but $r = \frac{1}{|R|}$. If there are a lot of relevant data elements ($|R| \gg 1$), then recall r will be very small ($r \ll 1$).

Both of these extreme ends are not desirable — a good query processor has to balance both criteria and deliver a result set that has both a high recall and a high precision score. Theoretically this is easy, if the result set contains just the relevant data elements: if $Q = R$ then $r = \frac{1}{1}$ and also $p = \frac{1}{1}$. In practice, especially when searching in sets containing complex data, such result sets are difficult to calculate.

A well-known example for this problem are World Wide Web search engines, such as Google [3]. Users of these search engines tend to enter only a few keywords, and expect to get a list of Web pages that is both complete

(has a high recall score) and pure (has a high precision score). Early search engines used simply a full-text index of known Web pages created offline by an indexer or robot process to achieve this goal [63].

However, such a design is not sufficient, because neither completeness nor purity can be guaranteed. A result list based solely on matching the keywords entered with the full-text index will not contain relevant Web pages that refer to their topic using synonyms of the keywords entered — thus the recall score might be actually quite low. Also, if the keywords have homonyms, then the result list might contain quite a lot of pages irrelevant to the user – thus the precision score might be low.

Moreover, the result set containing the relevant data elements often is very large and some data elements in such a large result set tend to be more relevant than others. The solution is to assign a relevance score to each data element by means of a relevance metric.

Definition 4.2 (Relevance) *Relevance is a scoring function* $r : T \rightarrow \{0 \ldots 1\}$ *that assigns a value in the range between 0 and 1 to each existing data element* $e \in T$*.*

Given a relevance function r, the result set R of all relevant data elements can be defined as follows: a threshold value t has to be chosen, such that $R = \{e \in T \mid r(e) \geq t\}$. A common special case arises when the relevance function is defined as $r :\rightarrow \{0, 1\}$ (that will assign either the value 0 or 1 to each data element) and $t = 1$. In that case, a data element either belongs to the result set or not, and no further distinction is made.

Thus, we assume that for each query q, there is a corresponding relevance function r, which in turn defines the result set R.

We can now apply the notions of recall, precision and relevance to the Relevance Assessment service.

The Relevance Assessment service constantly receives awareness events — these events, or more precisely the set of all possible events can be conceived as the data elements set T. The Relevance Assessment service has to decide for each incoming event e and each user u, whether $e \in R_u \subseteq T$, where R_u is the set of all awareness events relevant for the user u of the awareness system.

Therefore, we can apply the definition of recall and precision to awareness systems and describe the goal of the Relevance Assessment service as the maximization of both values for all users of the system: a metric for that goal could be for example the sum of all precision and recall values for all users, $\sum_{i=1}^{n}(r_{u_i} + p_{u_i})$, where n is the number of users.

The decision whether a particular event e is relevant for the user u ($e \in R_u \subseteq T$ or $e \notin R_u \subseteq T$) needs to be based on an individual interest profile —

that profile can be expressed as a relevance function r. However, there is one significant difference to traditional information retrieval: because awareness systems deal with dynamic rather than static data elements, the relevance of an awareness data element can also depend on the current time (e.g. an event that occured two weeks ago may not be as relevant as a similar event that occured an hour ago). Consequently, the relevance definition introduced above needs to be extended by the notion of time.

Definition 4.3 (Dynamic Relevance) *Dynamic Relevance is a scoring function* $r : T \times \Theta \rightarrow \{0 \ldots 1\}$ *that assigns a value in the range between 0 and 1 to each existing data element* $e \in T$ *considering the current time* $t \in \Theta$.

With these terms, we can describe more precisely what happens, when a `RelevanceService` object receives an event from a `CollectionService` (Figure 4.9).

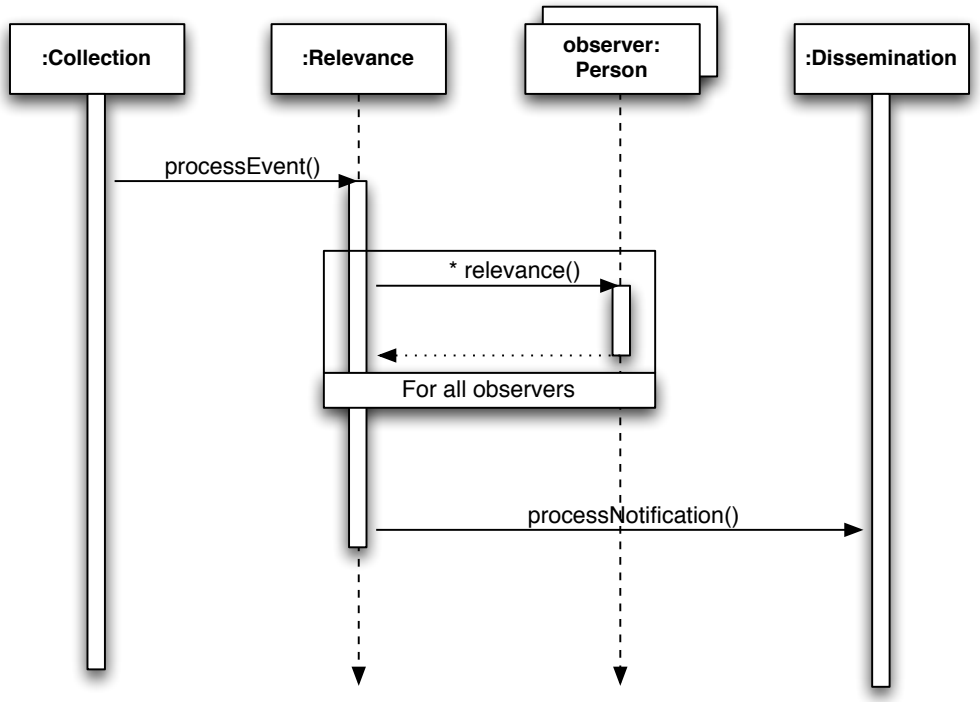

Figure 4.9: The interactions between the Collection Service, the Relevance Service and the Dissemination Service of an Awareness System

The `RelevanceService` object needs to know about both the individual relevance functions for all users of the system (the potential observers) as well

as a global threshold value. The relevance function, being an individual property of the user, can be easily modelled as a method `relevance(AwarenessEvent e)` in a class `Person` that takes an `AwarenessEvent` instance e and returns a floating point number in the range from 0 to 1. The current time can be usually accessed using a global function and does not need to be passed as a parameter.

The `RelevanceService` object needs to calculate the individual relevance functions using the incoming `AwarenessEvent` as the input parameter for all observers — if the value calculated by the individual relevance function exceeds the global threshold value, than an `AwarenessNotification` is created and handed over to the `Dissemination` service.

An `AwarenessNotification` consists of the observer's user id, the received `AwarenessEvent`, and the value calculated by the observer's `relevance()` method (Figure 4.10). The `RelevanceService` object needs to create one `AwarenessNotification` object for each user whose `relevance()` method returns a value above the global threshold. Thus each `AwarenessEvent` received by the `RelevanceService` object may result in a different number of created `AwarenessNotifications`: between zero and the number of the system's users, if everyone is interested.

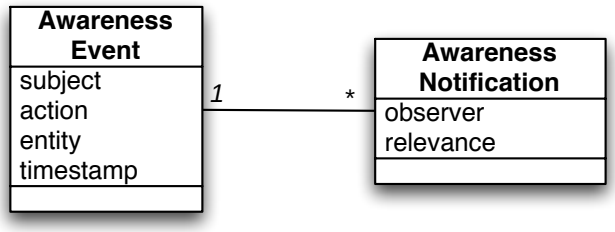

Figure 4.10: The `RelevanceService` object creates a distinct `AwarenessNotification` for each interested observer.

The `RelevanceService` object hands `AwarenessNotifications` over to the `Dissemination` service as soon as they are created. So, any buffering and storage for later delivery falls into the responsibility of the `Dissemination` service.

The calculation of all the individual relevance functions may take significant time, because the `RelevanceService` object needs to loop over all users and call their `relevance()` methods. Depending on how complex the actual calculations of these methods are, this mechanism may cause a delay. Thus, as previously mentioned, an appropriate buffering mechanism is also needed between the `Collection` and the `RelevanceService` object.

4.4.2 Calculating Relevance

We have not yet answered the question on how the Relevance Assessment service is able to calculate a meaningful individual relevance function. The choice of the right relevance function is a critical decision that affects the both the usability and effectiveness of the awareness system. Unfortunately, (as we will later see in this section) there is no obvious way to calculate that function. To address this problem, we propose a flexible approach that allows us to experiment with different relevance functions[2].

To simplify the discussion, we want to assume that users express their interest in certain events by hand, creating objects of type `Interest`.

We start with a simple notion of interest and than gradually extend it to include the concepts of importance weight and precision. For each step, we will discuss formulae that can be used to calculate the relevance function.

Formally, an interest I can be described as being a subset $I \subseteq T$, where T is the set of all possible awareness events. A particular interest can therefore be represented by an object of type `Interest` (Figure 4.11), that describes a subset of awareness events.

An observer can create an arbitrary number (e.g. n) of `Interest` objects. Consequently, there are n sets I_i that correspond to those `Interest` objects and that together form the set $R_u = \bigcup_{i=1}^{n} I_i$ that contains all `AwarenessEvents` relevant to the user u.

Each time the Relevance Assessment service calculates the individual relevance function, it calls the observer's `relevance()` method, that in turn, would go over all the `Interest` objects of that observer to find out if there are `Interest` objects that correspond to the incoming `AwarenessEvent`.

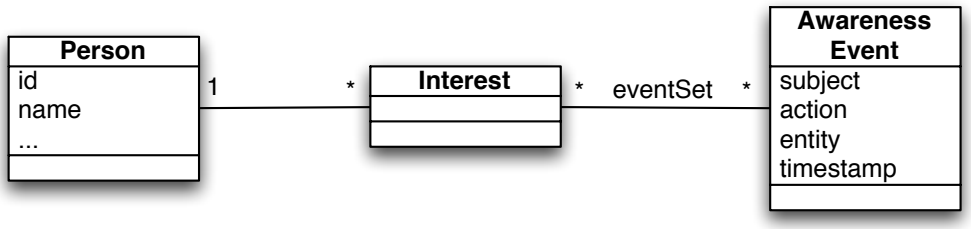

Figure 4.11: A Person can create an arbitrary number of Interest objects.

A simple `relevance()` method could calculate the relevance $r : T \rightarrow \{0 \ldots 1\}$ by simply returning the value 0 if no `Interest` object matched and

[2]Chapter 6 documents the results of an empirical evaluation of the relevance functions presented in this section.

the value of 1 if there was a match. This approach can be described by the following simple formula:

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ 1 & : \quad e \in R_u \end{cases} \tag{4.1}$$

However, this results in a situation where the receiver of the resulting notification (the observer) will not be able to distinguish between events that just barely touched his interests (i.e. matching just one `Interest` object) and events that touch all his interests (i.e. matching all `Interest` objects). Thus the above formula works only if the interests do not overlap.

If the interests overlap ($\bigcap_{i=1}^{n} I_i \neq \emptyset$), we can assign values from the range between 0 and 1 depending on the number of `Interest` objects that matched. This leads to an alternative formula, where we calculate a fraction by dividing the number of matched interests by the number of all stated interests:

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ \frac{|\{I_1 \leq i \leq n \subseteq R_u : e \in I_i\}|}{n} & : \quad e \in R_u \end{cases} \tag{4.2}$$

With this formula, we get $r(e) = 0$ if no interest has been matched at all and we still can get $r(e) = 1$, if and only if all n interests match the event. If only one interest matches, than we will get $r(e) = \frac{1}{n}$, which can be very small if $n \gg 1$. We get some value in the range between $\frac{1}{n}$ and n if more than one interest has matched.

This seems to be intuitive if the interests do overlap , but somewhat fails if they do not: in that case we will never get the value of 1, in fact, we will never get anything other than either 0 or $\frac{1}{n}$.

Any decision between our first (Formula 4.1) and our second formula (Formula 4.2) would be arbitrary at this point. Only if we knew in advance whether our awareness system would support overlapping or non-overlapping interests could we make the decision. Thus, for the general case it becomes clear that an awareness system should be rather agnostic to the algorithm used to combine multiple matched interests into one relevance value.

This issue becomes even more difficult if we decide to include the notion of interests that are more important than other interests in our application domain. We call them importance weighted interests. Those interests are an extension to the simple notion of interests introduced above, albeit a simple one: they include an additional attribute, `importance` (Figure 4.12). Like relevance, importance can be any value in the range between 0 and 1.

In normal language use, *importance* and *relevance* are similar concepts, often they are even used as synonyms. In our context, however, those two terms are used deliberately to distinguish two different concepts: relevance is

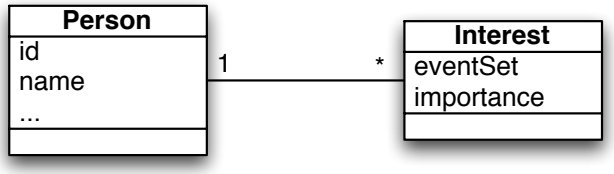

Figure 4.12: The notion of interest can be extended to include an importance weight.

a calculated value that measures how useful an event might be for a particular user, while importance is a user specified value that describes how important future events of a certain type will be for the user. Obviously, importance of interests can serve as an input value to the calculation of relevance.

When calculating the relevance using importance weighted interests, the Relevance Assessment service would still call the observer's `relevance()` method, that in turn would go over all the `Interest` objects of that user to find out if there are `Interest` objects that correspond to the incoming event.

While we still need to make the distinction between the overlapping interests case and the non-overlapping interests case, we now have to take the importance values into account when trying to combine multiple matched interests into one relevance value.

In the simple, non-overlapping case, we can extend our first simple formula (Formula 4.1) to return the importance value of the matched interest instead of the value 1. In the trivial case, where the event did not match any interests at all, we still can safely return the value of 0:

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ importance(I_i) & : \quad e \in I_i \subseteq R_u \end{cases} \tag{4.3}$$

In the overlapping case, an event e can match a number (e.g. m) of interests I_i, with $1 \leq i \leq m$. There is no "natural" way to combine the resulting values $importance(I_i)$ into one — rather one can think of several competing approaches.

First, we could take the average of the importance values as the relevance return value. This approach can be easily expressed in a formula:

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ \frac{\sum_{i=1}^{m} importance(I_i)}{m} & : \quad e \in I_i \subseteq R_u \end{cases} \tag{4.4}$$

However, it appears a bit non-intuitive to degrade the relevance of an

event because it matched not just one high importance interest but also a couple of low importance ones.

Another approach would be to take the maximum of the importance values matched. This method is even simpler to understand and to calculate. Also, the relevance of an event with a high importance interest would not be degraded by a couple of low importance interests that also matched.

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ \max(importance(I_i)) & : \quad e \in I_i \subseteq R_u \end{cases} \tag{4.5}$$

Still, neither Formula 4.4 nor Formula 4.5 allow the importance values to cumulate. Intuitively, it appears obvious that those events that match several of one's interests should have a higher relevance than those that match just one. Unfortunately, simply adding up the importance values of the matched events can result in values larger than 1 and therefore not fitting in the range provided for relevance. We could, however, accumulate anyway and simply cut off the sum at the value of 1. This would lead to the following formula:

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ \min(\sum_{i=1}^{m} importance(I_i), 1) & : \quad e \in I_i \subseteq R_u \end{cases} \tag{4.6}$$

However, when using this cumulative approach, the calculations can become more complicated and harder to understand by the users of the system. as a result, the usability might suffer. Usability as a criterion could favor simpler formulae, like Formula 4.5 that uses the maximum approach.

Another decision has to be made regarding the weight of focused `Interest` objects vs. broad `Interest` objects. An example for a very focused `Interest` object is an `Interest` that matches exactly one `AwarenessEvent`, i.e. the subset of all events that the `Interest` represents contains only one element. Broad `Interest` objects match a larger number of events — a very broad example is an `Interest` objects that matches all possilble events.

The point can be made that there is really no difference between focused `Interest` objects and broad `Interest` objects and there is no need to handle them differently. If the user has stated his interest with a broad Interest object, than he is supposed to know what he is doing.

However, `Interest` objects can be considered as queries on the set of all possible events. In practice, focused queries, or Interest objects respectively, can be expected to have overall a higher precision score than broad queries. This would support giving higher weight to focused queries.

Still, it is difficult to decide how much one should favour the focused `Interest` objects in comparsion to the broad ones. Given that the higher

weight assigned to focused Interest objects is justified by their higher precision, the weight of the broad queries could be only degraded if their precision would be known.

We could extend our notion of Interest objects to include such a measure. Basically, an `Interest` object would need to include just one additional attribute: `precision`. To conform to the precision definition in the previous section, we postulate that the `precision` attribute should take values from the range between 0 and 1.

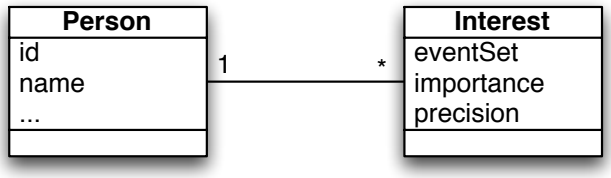

Figure 4.13: `precision` is another useful attribute for the `Interest` class.

Unfortunately, the `precision` attribute adds even more complexity to the already quite complicated problem of calculating the relevance function.

To simplify, the functionality of the relevance method can be split into two methods. The main method calculates the relevance as described so far: it scans all observers (of class `Person`) and all their interests and finds those interests that match the event for each observer.

After finishing the matching process for a given observer, it does not calculate the return value (using the importance and precision values from the interests found) by itself, but delegates this calculation to a helper method.

This helper method takes an array of importance value as its parameter and can calculate the relevance using any algorithm available. Simple helper methods could use algorithms employing either the average (Formula 4.4) or the maximum (Formula 4.5) formula. Should these formulae turn out not to be sufficient for the chosen problem domain, the implementation of the helper method can be replaced or a new helper method can be called.

The replacement can be done either by modifying the method directly, or by subclassing the `Person` class, and overriding the helper method with a new implementation. An even more flexible approach could incorporate a strategy pattern, which allows switching the helper algorithms at runtime (Figure 4.14).

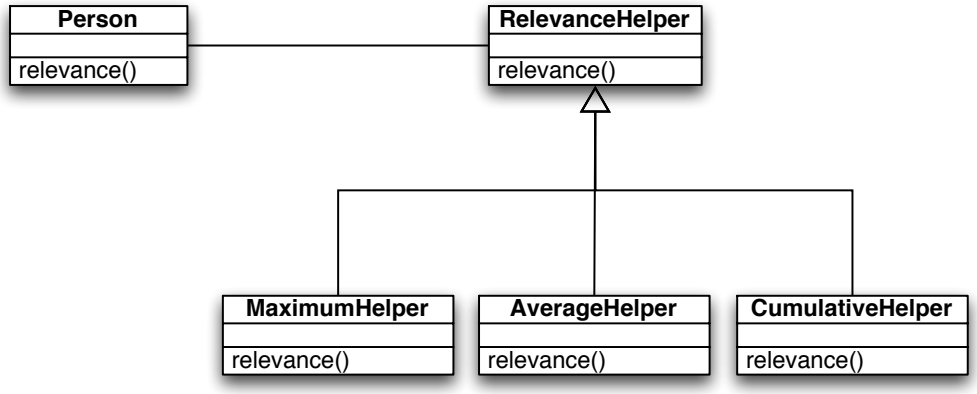

Figure 4.14: Selecting different interest combination algorithms with the strategy pattern.

4.4.3 Automatic Focus Update

While manual configurability is important, and often works better than automated approaches, it also imposes a burden on the users of the awareness system. When their work focus changes, they have to remember to change their interest profiles in the awareness system. No matter how intuitive and easy to use a user interface for doing that might be, the users cannot be really expected to do that constantly.

Thus, some way of automatically creating `Interest` objects has to be provided. Assuming that the awareness system obeys the reciprocity principle introduced in Chapter 2 (as one of the privacy principles for awareness systems), automatic user profile generation can be done through an analysis of a user's own actions which are already being fed into the system (Figure 4.15).

Also, this is not a violation of the privacy principles as this analysis is used to create a observer's own profiles and is not distributed to other users of the system.

A model for an observer's current interests can be derived from the concept of a user's *focus* described by Rodden [69]. Using spatial methaphors as a starting point Rodden formulated a general model consisting of a *shared space* populated by "objects". This shared space is used by a community of users for which the proximity to those "objects" can be measured.

This general model can be applied to both a physical space with its geometric properties as well as to a virtual world, where the notion of proximity has yet to be defined. It is similar to our `Workspace` model that is populated by `Person` and `Entity` instances.

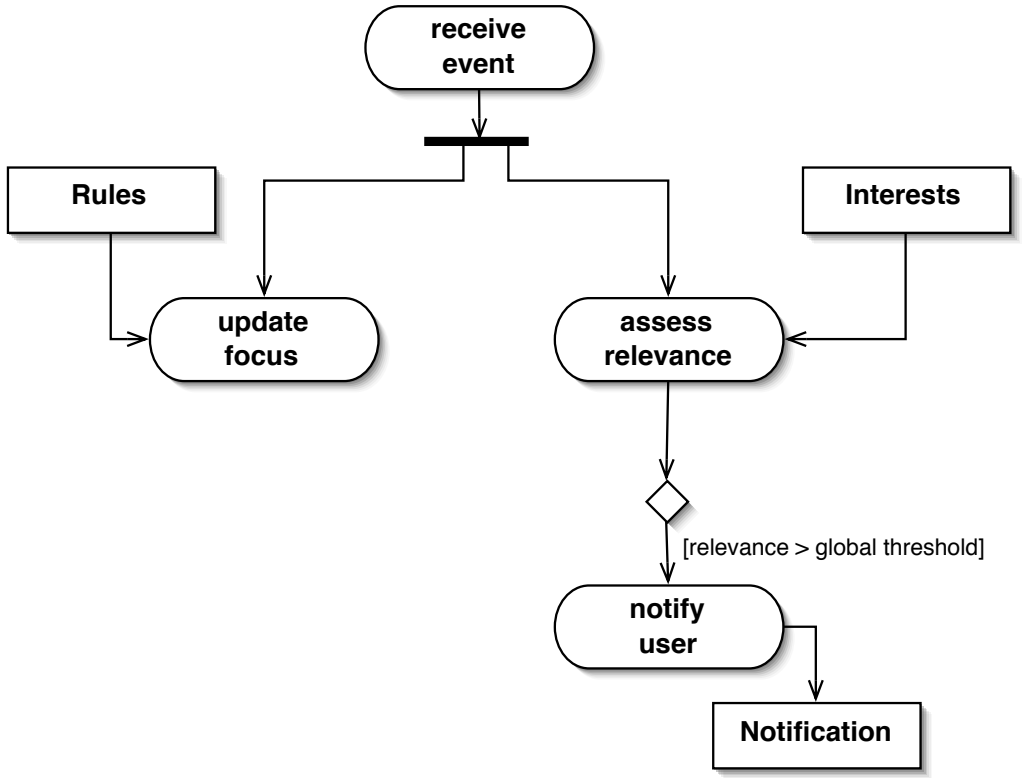

Figure 4.15: Automatic user profile generation can be done through an analysis of a user's own actions which are already being fed into the system.

A *shared space* SS is defined by Rodden as a pair $SS = (U, O)$ with $U \cap O = \emptyset$, where U is the set of all users and O is the set of all objects. Then, the *presence position* of a user is described as a pair $P(o, A)$ where $o \in O$ and $A \in \mathcal{P}O$ (where $\mathcal{P}O$ denotes the powerset of O). The *presence position* consists of the particular object the user is focusing on and a set A of objects adjacent to that object. The set of all possible presence positions (the presence space) is $PS = \{P(o, A) \mid o \in O \wedge A \in \mathcal{P}O\}$.

Given these definitions, focus can be defined easily as a function that maps a particular user to presence positions:

Definition 4.4 (Focus (simplified from [69])) *The user's focus is the function $f : U \rightarrow \mathcal{P}(O)$. It maps each user to a set of positions in the presence space. $f(u)$ is the set of presence positions that the user u can see.*

If an awareness system knows about a user's current focus, than it can, for example, assume that all awareness data regarding the contents of the focus

is relevant to the user and may be therefore forwarded to that user increasing both the recall as well as the precision of the awareness data received by that user. The relevance value assigned to such data elements could be as high as 1.

Also, as a user can change the focus over time, an awareness system should also assume that awareness data regarding the contents of past foci is also relevant to that user. However, the system should degrade the relevance of the awareness data in old foci as time passes. Otherwise, a user's focus would become broader and broader over time and at some point would match to everything that happens in the workspace.

The awareness system has two ways to learn about the user's current focus. First, the user can explicitly create `Interest` objects to define the focus. Second, the system can observe the user's own `AwarenessEvent` objects and calculate the focus from those events. For example, the user can always explicitly include a certain file in his focus The system, when observing that the user is modifying some other file, can automatically include that file in the user's focus as well. In any case we assume that the already discussed set of all `Interest` objects R_u is an approximation of the user's focus.

To support automatic `Interest` object creation, the `RelevanceAssessment` service needs to update the focus of the user from whom the event originated (the subject), before an `AwarenessEvent` is passed to the `relevance()` methods of all observers. This can be modeled as a call to a method `updateFocus()` of the subject (of type `Person`). After updating this focus, relevance assessment can be continued as described in the previous section — with the `RelevanceService` object calling the `relevance()` method of all observers (Figure 4.16).

We assume that the subjects are not interested in their own actions, because they already should know what they are doing, without the help of the awareness system. There are two ways to prevent the delivery of such self-monitoring events to the subject: one is to make sure that no `Interest` objects match those events, another is to simply not call the `relevance()` method of the subject user in question. While the first method would make the high level processing simpler, it would also hide this assumption deep in the Interest objects.

The second way does not complicate the high level event processing that much: after updating the focus of the subject user, the `RelevanceService` object calls the `relevance()` method of all observers, except the one of the originator. This can be easily achieved by checking the object ids.

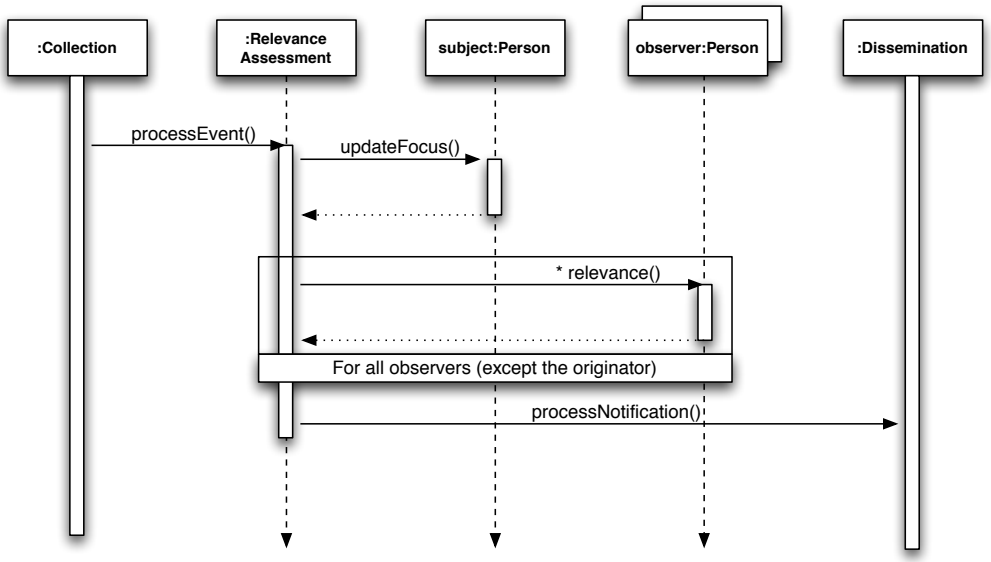

Figure 4.16: Before an awareness event is passed to the observers' relevance methods, the focus of the subject has to be updated

4.4.4 Entity Proximity

Next, we discuss the implications of introducing the notion of proximity into our model. Proximity is required to fully support Rodden's concept of *focus*, that was introduced in the previous section. In the physical world each user and each object have a location in the three-dimensional space, so an Euclidian distance can be calculated for any pair of users and entities.

We can use a fixed Euclidian distance value as a threshold for selecting entities belonging to the focus of a user, rather then specifying every single entity separately. The exact value of this fixed distance depends on the problem domain. We assume, that this value needs to be derived through empirical evaluation.

The real interest of the user may extend beyond the entities contained in the set that corresponds to the `Interest` object. It may include also entities that are "near" (within a given distance) the objects in the set. This can be included in the `Interest` class by defining an additional `distance` attribute (Figure 4.17).

This adds a small overhead to the process of matching an `AwarenessEvent` with an `Interest` object: rather then just checking whether the event is included in the set of events corresponding to the interest (the interest set), the `RelevanceService` object needs to check also the extended set including

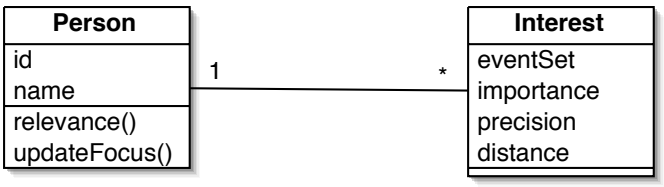

Figure 4.17: The actual interest of a user may include also objects that are "near" the entities in the set corresponding to the `Interest` object.

all events within the given distance of the interest set.

We could also allow for some degradation with distance. Entites near the center of the defining shape — those that are close to the interest set — can be thought of as being more important and having more focus as those near the edges of that shape. Thus awareness data concerning entities near the center of the focus shape could be assigned larger relevance values, while awareness data concerning entities near the edges of the shape could be assigned low ones.

This degradation can be achieved by a fixed formula that is common for all interests. The degradation needs to be significant — if it is not, than the `Interest` objects will match too many events and precision will suffer. The degradation formula could be implemented in the `Interest` class — however, being subject to experimental evolution, it should be better implemented outside the `Interest` class, e.g. in the RelevanceHelper method introduced earlier. To allow this, the `Interest` class needs to return a distance value rather then just a boolean decision, when asked whether an event matches the interest set or not.

In that case, the `distance` attribute becomes a `distance()` method: if there is a way to calculate the distance between two entities, than — given an `AwarenessEvent` — the `Interest` can calculate the distance between this event and the nearest member of its corresponding event set on the fly and return it to the caller.

Now the algorithm of the observer's `relevance()` method can be described as follows: that method contains a loop over all of the user's Interest objects, reads the importance and the precision attributes and calls their `distance()` methods. While looping, the importance, the precision and the distance values are stored for each `Interest` object in an array for later processing. That array is than passed over to the `RelevanceHelper` method to combine the values from the array into one value — the relevance score.

Calculating the distance for non-physical entities is a little more complicated, because there is usually no obvious distance metric. At first glance,

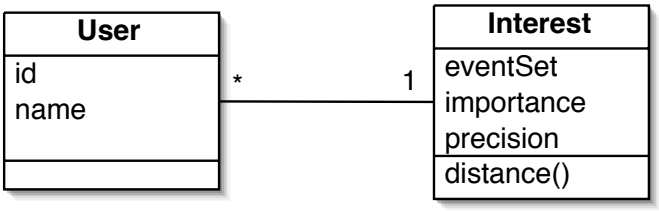

Figure 4.18: Implementing distance as a method rather than an attribute allows dynamic calculations.

if we do not want to loose generality, we can only resort to a simple binary metric that directly compares two objects. Using that kind of metric, the distance between arbitrary two objects is either 0 (if the two objects are identical) or 1 (if the two objects are two distinct objects).

Distance metrics are highly application domain dependent. Thus we delegate their calculation to an abstract class called `Distance`, that provides basically one method: `calculateDistance()` (Figure 4.19). This method takes two entities as its parameters and calculates the distance metric. Concrete subclasses of `Distance` may have global knowledge about the entity-entity distances (e.g. a distance based on the containment hierarchy of documents and folders ordered in a file system), or they may need to query the entities and/or the controlling applications for that information (e.g. a distance based on some notion of similarity that is calculated based on the contents of the documents).

For example, most shared applications used, e.g. network file services, usually impose a containment hierarchy on their objects. In a hierarchical file system, files can be contained in directories, and these directories can also be contained in other directories. Thus, a distance metric based on the hierarchy can be used in such circumstances.

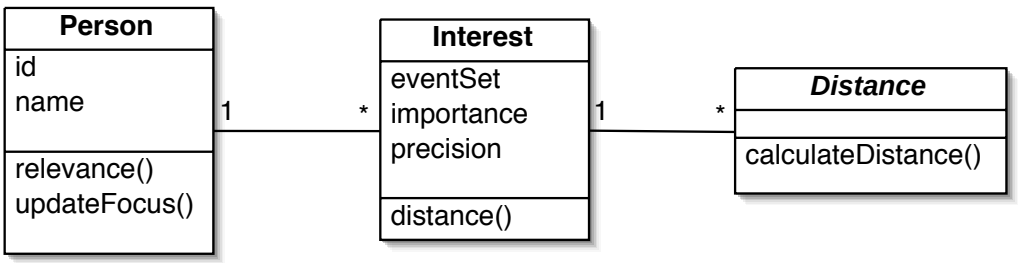

Figure 4.19: The abstract `Distance` class encapsulates the complexity of calculating the distance between two entities.

An awareness system should support a number of different `Distance` objects, corresponding to different `Entity` subclasses. Also, these objects should not be statically bound to the `Interests` at compile-time, but rather should be selected dynamically. A mechanism for dynamically loading code at runtime should be used to allow extensibility without having to recompile. The `distance()` method of an `Interest` should then loop over all available `Distance` objects and check the distance returned by the `calculateDistance()` method of each model. `Distance` objects that do not support a given `Entity` pair can simply return a value corresponding to infinity.

4.4.5 Describing Awareness Event Sets

So far we have presented an entity-centric view of interests. However, focus and interests are multi-dimensional. While users might focus their interests on certain entities or sets of entities, they might also focus on certain other users or sets of users (e.g. teams). They also might want to express interest (i.e. focus) in certain types of actions.

`AwarenessEvent` objects can be thought of as points or vectors in a multi-dimensional *awareness space*, with their attributes corresponding to the dimensions of that space. `Interest` objects can be then interpreted — analog to the general definition in the three-dimensional physical object space — as arbitrary shapes in this space.

To specify an `Interest` object we need to find a way to specify the corresponding set of `AwarenessEvent` objects. The simplest way to specify such a set is to enumerate all its member elements. Unfortunately this method is only feasible for small sets.

For large `AwarenessEvent` sets other methods have to used. A promising approach is to use a formal language.

The set of all legal values for any given attribute or dimension of an `AwarenessEvent` (in our workspace model either user, object or operation) is a formal language. Thus, we can use grammars or, if the language is regular, regular expressions to specify all the elements of an `AwarenessEvent` set.

Although event definition languages have been defined in the past (e.g. generalized path expressions for the BEE system [15]), they might be too complicated for end users — who should be able to create their own `Interest` objects manually — but regular expressions are something that many end users might be able to cope with.

For example, in many applications, hierarchical name spaces already exist. Object identifiers can then be defined easily as paths describing the unambigous object position in the hierarchy, rather then or in addition to abstract identifiers (e.g. file paths in a shared file system). If this is the case,

then users are able to describe subhierarchies with simple regular expressions (e.g. /cvs/abx/*).

Obviously, the namespace for the hierarchy has to be clearly defined and documented, so the end users either can be expected to know the names of the nodes in the hierarchy or the user interface of the awareness system can provide hints about legal node names.

It remains to be shown by means of experiments whether regular expressions are expressive enough for a particular problem domain.

Regular expression do not allow to describe arbitrary sets, but are rather restricted to regular sets which are by themselves only a small subset of the set of all possible sets. We can expect that the regular expressions are only an approximation of the actual set.

Thus, the precision value introduced earlier should be a property of the regular expressions describing the individual attributes, rather then being an attribute of the Interest object. The `relevance()` method should be changed accordingly.

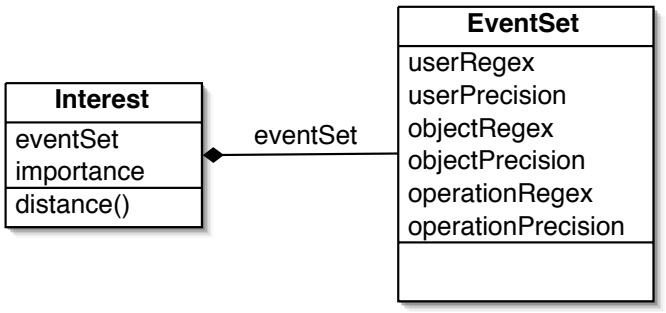

Figure 4.20: Regular expressions can be used to describe a set of `AwarenessEvent` objects

4.4.6 Relevance Assessment Summary

We have shown that relevance assessment is a special case of the general information retrieval problem, where a relevance function needs to be calculated for each incoming event.

We have presented several approaches to calculate a relevance function. Our focus is to use a function that can be intuitively understood by the user. Otherwise users get confused on the rationale why a certain event received a particular relevance score.

For this reason, we have proposed to using formulae that employ simple principles such as taking the average or the maximum that can be easily

understood by end users. Also, we have proposed to take special care to make the relevance algorithm easily modifiable in terms of the actual formula, to allow experimentation.

We have shown, that a user's interest profile (focus) can be easily updated automatically. We have also discussed how object proximity can be taken into account and how event sets can be described to define a user's interest profile. In particular, the last two points require careful experimentation to take usability into account.

4.5 Dissemination of Awareness Information

After an event has been found worth forwarding to an observer, the Relevance Assessment service creates an Awareness Notification, packaging a copy of the event with a relevance value and an id of the user to be notified (Figure 4.10). Immediately after its creation, this Awareness Notification is handed over to the Dissemination service for further processing.

We assume that the observers have clients that can be used to display notifications. Those clients can be either active or inactive from the awareness systems point of view. Each observer can have one or more clients active at the same time, e.g. a GUI client on his workstation and an ambient client on the wall of his room.

Electronic mail (email) is another possibility to disseminate electronic awareness information. In fact, certain existing systems like CVS can be configured to automatically disseminate awareness information (e.g. commit logs) to interested parties via email. However, experience shows [38] that email users are often very dissatisfied with a large part of mass emails and electronic newsletters they receive. They feel that the email they receive is irrelevant and distracts them from their real work . Email may be therefore not the best method for dissemination of awareness information.

This observation is consistent with the insights from the Media Richness theory. Media Richness has been used in the past to explain that communication media has to be wisely chosen to fit the communication task and a communication medium — such as email — that works well for certain tasks can fail when used for other tasks [22].

In the past, researchers have proposed a wide range of media to present awareness information. To be able to freely choose from the available presentation media, an awareness systems needs to be agnostic to the methods and display devices. The Dissemination service needs to provide the neccessary abstraction.

It needs to provide a way to dynamically connect various types of displays

to the awareness systems. It can be thought of as mirroring the functionality of the Collection Service, but rather then allowing the connection of devices providing input into the system, it allows the connection of devices providing output.

In addition, the Dissemination Service needs separate storage facilities as a precaution for presentation devices that can go offline. In that case, the Dissemination Service might need to store the Awareness Notifications destined for the offline device until it goes online again.

The awareness information stored might become outdated and in some cases can be safely discarded, though. The time interval between receiving awareness information and discarding it is highly application domain dependent. Such an notification expiration feature can be implemented by storing a expiration time interval attribute in the Interest object.

The semantics of the expiration feature have to be carefully chosen. The expiration time can mean that the notifications are discarded after a certain time interval has elapsed no matter whether they could be delivered to the user or not. It can also mean that the notifications are not shown in display client after the time interval has elapsed. However, the latter is better implemented by a separate, client specific preference that the user can set individually.

The various user presentation clients can be either connected or disconnected — if they are connected to the server component of the Dissemination Service (user is online), then the Awareness Notifications can be delivered immediately after being received from the Relevance Assessment service. If no client for the specific user is connected (user is offline), then the Dissemination service needs to maintain the Awareness Notifications in a buffer, either until they expire or until a client connects.

If a client connects to the Dissemination Service, then it needs to check whether there are yet undelivered Awareness Notifications for the particular user — if there are, then it sends them to the newly connected client.

Whether the Awareness Notfications can be deleted from the buffer after being send to a client has yet to be decided. There are two main possibilities: either we decide that delivering an Awareness Notification to *some* client of the user is sufficient, or we decide that we need to deliver Awareness Notifications to *all* clients of the user. In the former case, the Dissemination Service can delete a notification from the buffer as soon as it could be send to some client, in the latter case the Dissemination needs to keep track of how many clients a user has available.

We consider the clients to be a part of the Dissemination service. Some clients might maintain their own remote Awareness Notification management. However, some clients might prefer to store the received Notifications

on the server side.

This server-side storage allows users to use different clients on different systems to view the same set of received Awareness Notifications. Thus, the Dissemination service can provide an optional interface for clients that want to store the received Awareness Notifications on the server side[3].

4.6 Summary

With the collection, relevance assessment and dissemination, we have identified the key functional areas which need to be addressed by any awareness system. We have discussed the issues that need to be resolved when designing an awareness system and proposed solutions into a generic (problem domain independent) awareness framework.

The activities in an Awareness System are triggered when an awareness events is received by the Collection service. After receiving the event, the Relevance Assessment service takes care of updating the originator's focus and of assessing the relevance of the incoming event for all other users of the system. Every time the resulting individual relevance exceeds the global threshold, a notification is created and delivered to the user via the Dissemination service.

The design of components that provide these services is highly dependent on the application domain, however, and we have listed many of these dependencies during the course of our discussion, including:

- domain specific user sensors, application sensors or entity sensors,
- a domain specific relevance function,
- domain specific distance metrics for calculating event proximity.

In the next chapter we present the actual architecture derived from the requirements presented so far and discuss our prototype implementation, AWARENESS BUILDER.

[3]This notion of server side storage is comparable to using IMAP rather than POP for Email.

Chapter 5

ABX: A Tool for Building Awareness in Distributed Software Development Projects

Surveys of the participants of distributed project courses conducted at TUM and CMU revealed that many of them were dissatisified with the level of awareness they had about other team's activites. In particular, in personal communication, the participants often expressed that the awareness of activities performed at the respective partner site was low, despite formal ways to communicate the results.

This observation and the theory described in the previous chapters motivated the development of a tool called AWARENESS BUILDER, which is described in this chapter. The goal of AWARENESS BUILDER is to introduce awareness — hence the name — in distributed software development projects.

AWARENESS BUILDER, or short ABX[1], can collect, rate, disseminate and visualize awareness information using existing software development tools as data sources. It provides a platform independent application sensor interface for data sources and includes two example implementations for such sensors: one for the software configuration management tool CVS and one for a custom communication infrastructure based on LOTUS NOTES.

We chose these two applications because they are the two cornerstones of the development and communication infrastructure used in the projects mentioned above. Also, both provide extension mechanisms, so application sensors can be effectively implemented.

We have implemented a traditional GUI-style user interface. We believe that little effort would be needed to allow ambient displays to be used for

[1] AWARENESS BUILDER for Mac OS X

presentation of awareness information using ABX — assuming a device driver and/or library that would allow to communicate with the device in question. In addition to the presentation of awareness information, the ABX client GUI provides a way to interactively set and modify one's interest preferences.

Having the deployment environment in mind, we decided to implement ABX using the Objective-C programming language and Apple Computer's object-oriented framework COCOA. While this choice limits portability of the server component and the client, it also allowed to speed up the development significantly. It does not limit the platform choices for the sensors, due to the use of open and widely available communication protocols.

In this chapter we cover the design and the implementation decisions in the AWARENESS BUILDER system and discuss the rationale behind those decisions. In Section 5.1 we present the architecture of our system that we derived from the requirements presented in Chapter 4. Then we discuss the individual subsystems: the Collection subsystem (Section 5.2), the Relevance Assessment subsystem (Section 5.3) and the Dissemination subsystem (Section 5.4). We close this chapter with a discussion of an alternative approach that uses existing features of CVS and LOTUS NOTES (Section 5.5).

5.1 Architecture

5.1.1 Subsystem decomposition

In Chapter 4 we have described awareness systems in terms of three services that can now be mapped to subsystems: the collection subsystem, the relevance assessment subsystem and the dissemination subsystem (Figure 5.1).

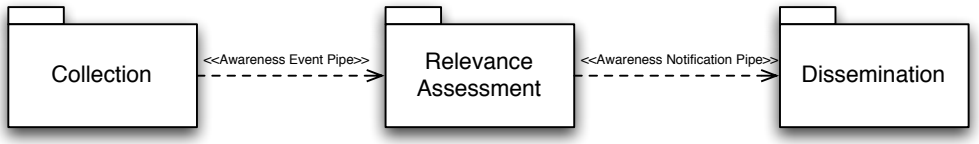

Figure 5.1: A typical awareness system can be modeled as a simple instance of the pipe and filters architecture style.

Because of the flow of awareness information a typical awareness system can be modeled as a simple instance of the pipe and filters architecture style [9]. In a pipe and filters architecture the first subsystem receives data from a set of inputs, does some processing and then sends the results to next

subsystem by means of a set of outputs. Each subsystem is called a filter and each filter is associated with other filters by means of pipes.

A common example of the pipe and filters architecture is the Unix-style command line language. Most commands are written such that they can read their input from one pipe and write their output to another. Simple syntax allows to chain together several commands with pipes such that the output of one program is fed to the next program as input. Complex scripts can be written using simple commands as building blocks.

The awareness architecture has three filter types, corresponding to the three services mentioned above. The Collection Service is driven by awareness events external to the system. It needs to be able to receive these awareness events from a variety of sources and transform them into a common representation. It hands over the awareness events to the Relevance Assessment service, which then decides how relevant they are for each user of the awareness system.

The Relevance Assessment service creates awareness notifications for those users, who need to be notified about an event. It uses the awareness notifications pipe to hand those notifications over to the dissemination service for delivery to the user.

5.1.2 Distribution and Hardware/Software Mapping

Each awareness system consists of a centralized component, the Awareness Server, that incorporates the relevance assessment service, as well as at least parts of the collection and the dissemination service.

Sensors collect awareness data from entities that in general case are not collocated with the Awareness Server. The Awareness Server usually runs as a long-running process, while the awareness data is often gathered in locations where the group members work. Thus, the collection service needs to be distributed: part of it needs to reside on the Awareness Server, parts of it will have to reside on a separate Awareness Sensor component.

Also, the relevant awareness information needs to be disseminated to ambient displays and/or via applications running on the computers that are used by the group members. Thus, the dissemination service needs to be distributed as well and parts of it resides on a separate Awareness Client component.

Due to the distribution of two out of three services the Awareness System is inherently distributed (Figure 5.2).

The Awareness Server implements the core functionality of our system. Figure 5.3 outlines the staged event processing including an (optional) archive event activity. This activity is not needed to fulfill the requirements of an

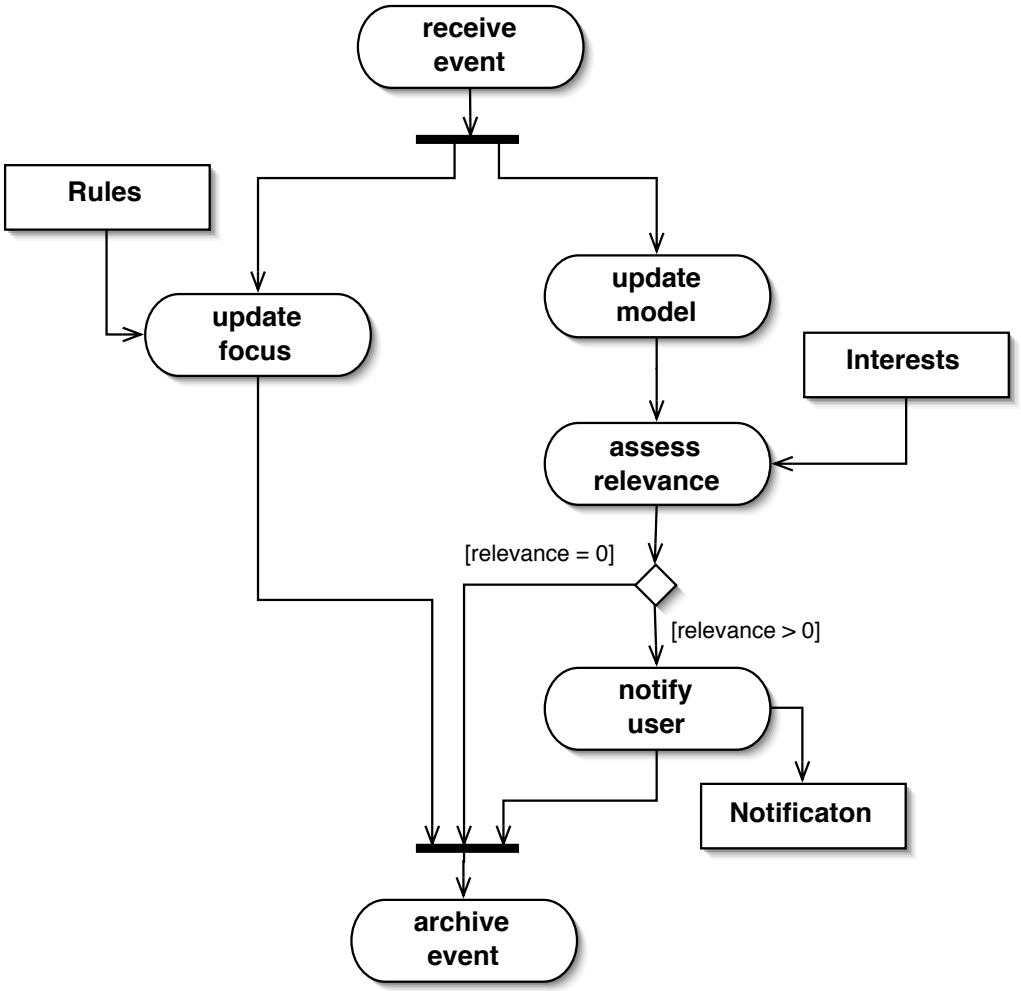

Figure 5.3: An outline of the activities involved in processing an ABXEvent.

respective applications. The XML-RPC and SOAP proxy sensors and ABXD (the ABX server) are separate server processes (implemented as daemons). We refer to the client application simply as the ABX client).

The Collection service is distributed between the two sensors, the two proxy servers and parts of ABXD. The Relevance Assessment service is almost completely contained within ABXD, with the client application AWARENESS BUILDER providing a user interface for configuration. Then, the Dissemination service resides partly within ABXD and partly within the client application.

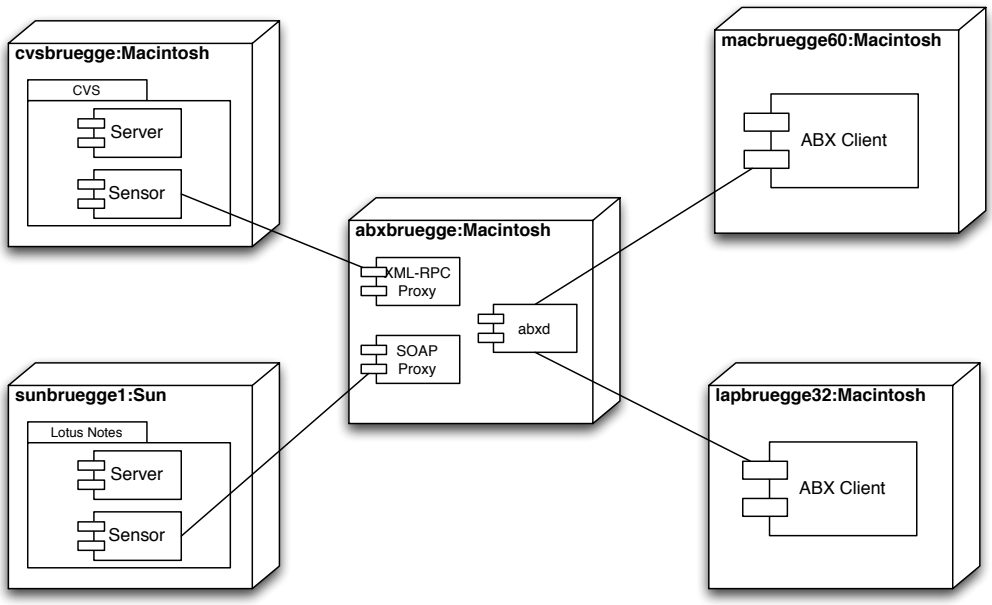

Figure 5.4: The ABX system consists of six components: the two application sensors, two proxy sensors, a server component and a client application.

5.2 Collection Subsystem

5.2.1 Workplace Model

ABX, is based on the workplace model introduced in Chapter 2 and refined in Chapter 4 that consists of the classes `Person`, `Application`, and `Entity`. We required that `AwarenessEvent` objects are created when users perform actions on entities.

In ABX' implementation, `Person`, `Application`, and `Entity` are each mapped to text strings. Consequently, the corresponding associations of the `AwarenessEvent` collapse to text attributes.

We represent users of the system, that is instances of the `Person` class, with a unique text that is consistent across all applications used. In our development and communication environment, a user's login name is a natural choice — it is a unique text string that follows the traditional UNIX conventions (e.g. consisting of less or equal than eight 7-bit ASCII characters).

Applications are represented with a unique name that follows the same convention as the user name. The two applications supported in our initial implementation, LOTUS NOTES and CVS are represented with the strings "notes" and "cvs" respectively.

Entities are private to the applications in the sense that users cannot manipulate them directly but rather have to use applications as proxies. Consequently, the entity identifiers are application dependent and may or may not be human readable. Still, those identifiers can also be represented as text strings.

For CVS entities (files), the path name was used as am unique identifer (e.g. /src/ABXFramework/ABXConstants.h). For LOTUS NOTES entities (postings on the message boards), the internal identifier can be used (e.g. 6C7A7C2007D6644FC1256A99005C8C55@@@@projects/tramp/coach.nsf/0/6c7a7c2007d6644fc1256a99005c8c55?OpenDocument@@TRAMP Infrastructure online!). The LOTUS NOTES can be easily parsed in the ABX client to derive the human readable name of the message board used (e.g. projects/tramp/coach.nsf) and the subject of the posting involved (e.g. "TRAMP Infrastructure online!").

Each application (LOTUS NOTES and CVS) in our deployment scenario acts upon exactly one class of entities, and there are no entities that can be manipulated by more than one application. Therefore, the applications impose a hierarchy on the entities, and the application identifier can be thought of as being part of the object identifier.

5.2.2 Awareness Sensors

In Chapter 4, we proposed the use of proxy sensors to allow for different protocols for communication between applications and the awareness system (Figure 4.7 and Figure 4.8). In ABX, we decided to use Distributed Objects (part of the COCOA framework) as the native collection protocol. In addition, we have implemented two proxy sensors: one proxy sensor that supports XML-RPC [77] and one proxy sensor that supports SOAP [80].

The XML-RPC protocol satisfies both the platform independence and the language independence requirements. Similar to traditional RPC, XML-RPC allows to invoke remote functions across networks. The remote functions are invoked by sending a function description (see below) that includes the function name and its parameters in a XML message over a HTTP connecton. The receiver of such a message parses the XML message, invokes the function locally and sends a XML message containing the result back to the caller.

```
<?xml version="1.0"?>
  <methodCall>
  <methodName>abx.processCommand</methodName>
  <params>
  <param>
```

```
    <value>
      <struct>
        <member>
          <name>user</name>
          <value><string>kobylins</string></value>
        </member>
        <member>
          <name>application</name>
          <value><string>cvs</string></value>
        </member>
        <member>
          <name>command</name>
          <value><string>commit</string></value>
        </member>
        <member>
          <name>object</name>
          <value><string>/src/ABXConstants.h</string></value>
        </member>
      </struct>
    </value>
  </param>
  </params>
</methodCall>
```

HTTP connections and XML message creation and parsing are supported on many platforms. Today, XML-RPC can be easily implemented in most environments and using a variety of programming languages: implementations are available for Perl, Python, Java, Frontier, C/C++, Lisp, PHP, Microsoft .NET, Rebol, Real Basic, Tcl, Delphi, WebObjects and Zope and others [1].

While XML-RPC provides a working solution, it has some limitations that have caused further development of the protocol. The result was the SOAP protocol: Simple Object Access Protocol. While the XML-RPC is strictly procedural, SOAP uses the object-oriented paradigm and can be used to pass objects over the network.

Historically, the ABX XML-RPC proxy sensor has been developed first. However, it became soon apparent that ready-to-use SOAP libraries make working with SOAP as easy as working with XML-RPC. After the SOAP proxy sensor has been implemented, the XML-RPC remained in place as an option for environments where the added overhead of SOAP is prohibitive.

Employing these two proxy sensors, awareness event creation has been implemented by modifying the two applications CVS and LOTUS NOTES we

chose to support. This way, subjects are not required to execute any additional awareness specific operations. Also, no complex monitoring sensors for the objects manipulated by these applications had to be implemented.

CVS provides built in support for invoking scripts after certain operations are executed on the CVS server, one of these operations being "commit". CVS passes the file path and number of additional data items as parameters to a small Perl script. This script (the CVS sensor proper), takes this arguments and creates a XML-RPC or a SOAP message in the appropriate format.

While CVS can been extended through scripting and configuration rather than coding, extending the custom LOTUS NOTES application was more demanding. The custom parts of the communication infrastructure have been written in LotusScript. Unfortunately, LotusScript supports neither XML-RPC nor SOAP directly. To avoid implementing yet another proxy sensor, LotusScript had to be extended. We used a portable SOAP library called GSOAP [4] to create new LotusScript methods and could then write an embedded awareness sensor in LotusScript.

Our original goal was to capture both "read" and "write" type of operations for both applications. While we achieved this goal for LOTUS NOTES, we only implemented support for the CVS "commit" operation.

The reason was that there is no easy way to track "checkout" operations in CVS. The usual way to use CVS involves checking out the current state of repository (a working copy) and then modifying files. CVS does not require a special operation before editing a file, like some other software configuration management systems do. Instead, it checks for conflicting changes when the user tries to commit his changes back to the repository.

Thus, the application involved when a user starts editing a file is the editor rather than CVS. We decided against modifying an editor to support the creation of awareness events, because this would violate the publicity principle (editor related actions are not public). Also, in our environment, we leave the choice of the editor to the individual user.

Another option was to change the behaviour of CVS through configuration: CVS can be configured in a way that encourages (but does not enforce) the use of an "edit" command before an editing operation. Users can use the "edit" command to notify the CVS server that they are about to start editing a file — and the server could be modified to create an awareness event every time they do.

Unfortunately, in practice, few users know about this command. The users can be encouraged to use the "edit" command by configuring CVS in a way that makes sure that all working copies are checked out with read permissions only. Using the "edit" command on a file has then the side effect of setting a file to read-write permissions.

We did not follow through on this option either. As we quickly found out by experimenting with this feature, users would either use the operating systems commands to change the file permissions (and CVS has no way of preventing this) or they would use the "edit" command on whole subdirectories rather than individual files.

We believe that the only way to reasonably use the "edit" feature would be to build support for it into the editors used. However, we have decided against modifying any one editor for reasons already stated.

5.2.3 Collection Behaviour

In ABX, event collection is triggered when a developer (subject) decides to commit a file to the CVS repository, or to read/write a message on one of the LOTUS NOTES message boards. In either case, one of the application sensors initiates an asynchronous XML-RPC or SOAP remote call to the corresponding proxy sensor (the return value is ignored).

The proxy sensor then creates an object of class `ABXEvent` (Figure 5.5). This class corresponds to the `AwarenessEvent` class that we defined during analysis. That object is immediately passed using Distributing Objects to the collection component of the AWARENESS BUILDER server ABXD where it is time stamped and put into a buffer where it awaits transfer to the server's relevance assessment component.

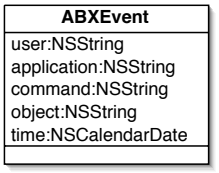

Figure 5.5: The collection process ends with the creation of an object of class `ABXEvent` (shown with their datatypes from the COCOA framework).

5.3 Relevance Assessment Subsystem

5.3.1 Calculating Relevance

When the Relevance Assessment component in ABX receives an `ABXEvent` object from a Collection component, it creates a number of `ABXNotification` objects for each user who is interested in the kind of events represented by

the particular `ABXEvent` object — that is, when the individually calculated relevance exceeds the global threshold.

This is achieved by a loop that scans all users known to the system, who are the potential recipients of yet to be created notifications. Within this loop, the relevance of the `ABXEvent` object is calculated (within the range between the value of 0 and 1) and a `ABXNotification` (Figure 5.6) object is created, if the calculated relevance value is greater than 0.

Each `ABXNotification` object includes this calculated relevance value as an attribute as well as the name of the recipient and the `ABXEvent` object. The created `ABXNotification` object is then passed to the dissemination service for delivery.

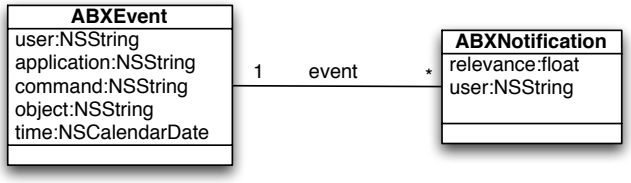

Figure 5.6: One `ABXEvent` can cause the creation of many `ABXNotifications`.

To calculate the relevance value, we use the INTEREST class introduced in Chapter 4. Each user has a number of corresponding `ABXInterest` (Figure 5.7) objects. To calculate the relevance value, a second loop is nested in the loop described above. For each user for whom the relevance value needs to be calculated, the inner loop iterates over the corresponding ABXInterest objects.

Figure 5.7: Each `ABXUser` has a number of corresponding ABXInterests.

The `ABXInterest` objects contain an attribute `eventSpec`, which describes a set of notifications using a simplified regular expression for each attribute of an `ABXEvent` — e.g. the expression */projects/tramp/coach.nsf/** would match all objects (messages) on this particular message board, while the expression *coaches/** would match all users assigned to the coaches team.

Objects of that type also contain a boolean method `isEventSpecEqual()` that takes an `ABXEvent` object and allows to check whether an `ABXEvent` object is contained in the set described by the attribute `eventSpec`.

The inner loop scans a user's `ABXInterest` objects and calls the `isEventSpecEqual()` method. It then combines the importance values from the `ABXInterest` objects that matched the `ABXEvent` object into the relevance value for the `ABXNotification` to be created.

To combine the importance values into one relevance value, the AWARENESS BUILDER system uses Formula 4.6 from Chapter 4:

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ \min(\sum_{i=1}^{m} importance(I_i), 1) & : \quad e \in I_i \subseteq R_u \end{cases}$$

Albeit not the simplest for the end user to understand (see discussion in Chapter 4), this formula allows importance values to cumulate and prevents the decline of relevance in the case where a high importance and several low importance interests match (we assume overlapping interests). This formula replaced the simple averaging formula (Formula 4.4) after the system was put into use in the ARENA project and users have voiced complaints about unexpected diminution (a thorough discussion of the deployment in the ARENA project can be found in the next chapter).

5.3.2 Event Proximity

We have simplified the description of the calculation process for the calculation of relevance for a given user in the previous section. The actual calculation process takes object and event proximity, as described in Chapter 4, into account.

Rather than calling the boolean `isEventSpecEqual()` method on an `ABXInterest` object and an `ABXEvent` we let an object of type `ABXModel` return the distance between the `eventSpec` of an `ABXInterest` and the `ABXEvent`.

The `ABXModel` object encapsulates the process of calculating the individual distances between the attributes of an eventSpec and the attributes of the `ABXEvent`, as well as the calculation of the resulting distance.

The `ABXModel` interprets an `ABXEvent` as a vector in the multi-dimensional space that is spanned by the attributes of the `ABXEvent` class. An `ABXInterest`, or more precisely, an eventSpec is a multi-dimensional shape in that space. The `ABXModel` calculates a distance vector with individual distance for the given dimension in each of it's components. Then, the `ABXModel` object sim-

ply calculates the average of the component values and returns the result as the distance between an `ABXEvent` and an `ABXInterest`.

Determination of a distance for a given dimension (e.g. user or object) is a non trivial task. Earlier work by Bürger [18] suggests to use a separate distance model for each dimension (that is for each attribute). We have found that we need to support more than that — e.g. some attributes need more than one model (e.g. the attribute OBJECT may require a different model for each application, see the discussion of the different models we implemented for CVS and LOTUS NOTES below), while others require no model at all (e.g. there is no obvious notion of proximity between applications, such as NOTES and CVS).

We decided to provide an open, flexible and dynamic interface that allows us to introduce new models at runtime. A model encapsulates the algorithms needed to calculate the distance between two values of an attribute. It is loaded at runtime by the mechanism provided by the COCOA framework and encapsulated in the `NSBundle` class of that framework.

We have implemented three different model subclasses as dynamic plug-ins to our system (Figure 5.8): an `ABXModel` subclass for the object attribute taking care of distances between CVS objects (`ABXCVSModel`), another `ABXModel` subclass for the attribute `object` taking care of distances between NOTES objects (`ABXNotesModel`) and an `ABXModel` subclass for the attribute `user` (`ABXPeopleModel`). Each of these models is able to detect whether it is able to do a meaningful distance calculation for a given event. It then returns either a nil value or an object that encapsulates the distance value for exactly one attribute.

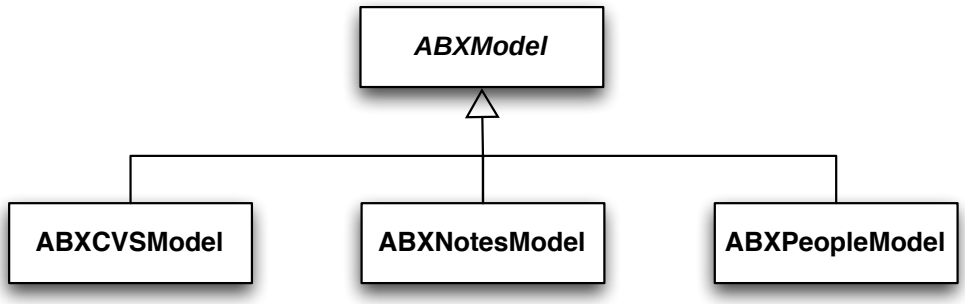

Figure 5.8: We have implemented three different model plug-ins, each being a subclass of `ABXModel`.

The case where more than one plug-in returns a distance value for an attribute is considered a configuration error, as we have decided to encap-

sulate the knowledge for each attribute in exactly one plug-in, to simplify implementation. In case of such an configuration error an error message is logged and the first distance value received is kept.

For example, the `ABXCVSModel` and the `ABXNotesModel` both calculate distances on the object attribute. As the `ABXCVSModel` does calculations on files in the CVS repository and the `ABXNotesModel` does calculations on postings in LOTUS NOTES, they cannot both return a distance value for the same event by design.

The `ABXCVSModel` supports source code files written either in Java or Objective-C, and is extensible to support other programming languages as well. To calculate the distance metric, it uses three steps: first, it calculates a metric based on the filesystem hierarchy, second, it calculates a second metric based on the inclusion hierarchy of the programming language used. In a third step, it combines these two metrics into one distance value that is returned.

In the filesystem hierarchy based metric, files that are in the same directory receive the highest score (value 1), while files that are in the same hierarchy but on different levels of that hierarchy receive a lesser score (value $\frac{1}{2^n}$, where n is the number of levels that are between the two files). Files that do not share a common hierarchy receive 0, the lowest possible score.

In the inclusion hierarchy based metric we have used a similar approach. Files that include one another (e.g. a class declaration and a class definition file) receive the highest score (again value 1), while files that include a one another indirectly receive a lesser score (again, value $\frac{1}{2^n}$, where n is the number of declaration files "between" the including and the included file). All other, unrelated files files receive 0, again the lowest possible score.

When combining these two values into one, after some initial testing within the project team (using the source code files of the ABX project itself), we decided to put more weight on the inclusion hierarchy based metric than on the filesystem hierarchy based metric. Our retrospective studies (see Chapter 6) showed that the following formula leads to a reasonable relevance distribution: $0.9 \cdot I + 0.1 \cdot F$, where I is the value calculated using the inclusion hierarchy metric and F is the value calculated by the filesystem hierarchy metric.

The `ABXNotesModel` supports messages posted on the custom LOTUS NOTES based message boards used in the surveyed projects. It calculates a simple hierarchy based metric that assigns the highest distance score to messages that belong to the same discussion thread and the lowest score to messages that are belong to different threads. We chose the value of 0.9 for the highest score (leaving the value of 1.0 reserved for matching a message with itself) and the value of 0 for the lowest score.

The `ABXPeopleModel` knows (it has to be configured manually) the projects organizational structure in terms of teams and people who belong to these teams. With this knowledge, it calculates a simple distance metric on the user attribute: people who belong to the same team receive the highest score (value 1), while people who belong to different teams receive the lowest score (value 0).

5.3.3 Manual Focus Update

In ABX, we have implemented relevance assessment that supports manual and automatic focus update.

As described in the previous section, the Relevance Assessment service relies on `ABXInterest` objects to calculate the actual relevance value for each user. We have provided a user interface that allows the user to create, modify and delete `ABXInterest` objects.

That user interface has been implemented in the ABX system within the AWARENESS BUILDER client. The user can view a list of all `ABXInterest` objects in a table view that displays the attributes Importance, User, Application, Command, Due Date and Object (Figure 5.9).

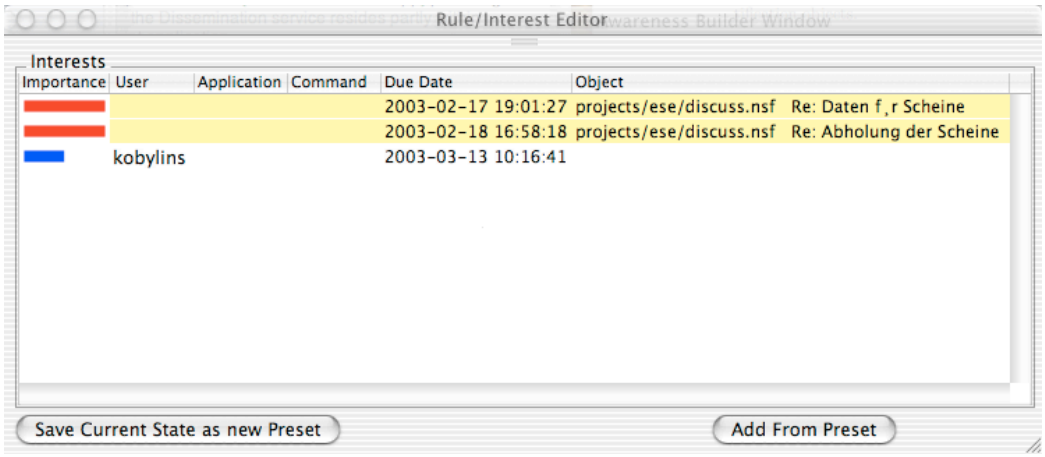

Figure 5.9: The AWARENESS BUILDER client allows each user to view his ABXInterest objects.

In Figure 5.9, we see three `ABXInterest objects`: two of them are highlighted[2] to signal that they have been generated automatically — we will

[2]yellow, which was chosen as the highlight color in the application, translates to a very light shade of grey in print

describe our implementation of the mechanism that allows to create interests automatically in the next subsection. Right now we focus on the `ABXInterest` object that has been created manually (the last one in the list): it matches all events that have been caused by actions by the user "kobylins", no matter which application he has used and which objects he manipulated.

The importance value of the `ABXInterest` object corresponding to this view has been manually set to 0.49. A graphical representation, a simple bar, is used to present that value to the user. The importance bars are displayed in different color values: this is due to a per user preference that stores the threshold between very important (red) and not so important (blue) events.

An individual threshold is different from the global threshold used in the Relevance Assessment service to decide whether a notification for a particular user should be created. The global threshold is set to 0 in ABX: this means that a notification is created if and only if the calculated individual importance value for an event exceeds 0. In addition, the user can set an individual threshold for what he considers most important — in our example, the user has set this individual threshold to 0.5.

Finally, the manually created `ABXInterest` object also contains a due date that marks the time when it expires and is automatically deleted. The user can also delete the ABXInterest objects manually by applying the menu command "Delete", or by pressing the Backspace-Key on the keyboard.

The user can sort the table entries by either attribute clicking on a column's title. Also, the user can resize the columns and rearrange them at will, by dragging them with the mouse.

To modify, or to create new ABXInterest objects, the Interest Inspector (Figure 5.10) has been implemented that allows to set the attributes of an existing or a newly created ABXInterest object.

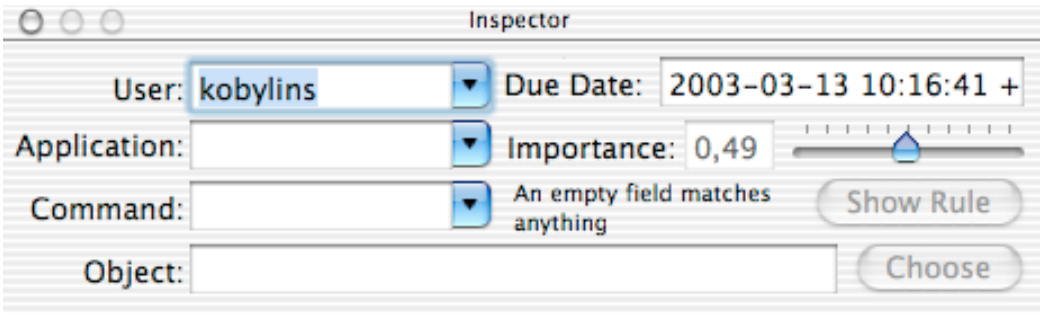

Figure 5.10: The attributes of an existing or a newly created ABXInterest object and the attributes of the associated ABXEvent object can be set in the Interest Inspector.

To promote end-user acceptance, the GUI includes a number of time saving features: the user can pick the value for most of the attributes from a pop-up list that contains the range of all valid values for the attribute. To set the user attribute, the user can bring up a pop-up list that contains the names of all project members (ABX supports one project per installation). Users can also simply start typing, and as soon as they typed enough characters to resolve any ambiguities, the value will be completed automatically. In our case, just typing "kob" would have been enough.

To simplify object selection, we have hardcoded some knowledge on how to pick objects for CVS and LOTUS NOTES into the client. When CVS is selected as the application, the "Choose" button activates and allows to pick a file or a directory from the working copy on the local filesystem (the user needs to set a preference with the path to the local working copy, though).

Also, when LOTUS NOTES is selected as the application, a pop-up menu can be brought up allowing the selection of one of the project message boards. Unfortunately, implementing the ability to select individual messages or message threads proved to be far more difficult, and was not implemented.

5.3.4 Automatic Focus Update

In the previous section we described how users can create, modify and delete `ABXInterest` objects and thus create a description of their current focus. However, when their focus changes, they need to manually update this description. The only automatic focus mechanism so far is the due date that prevents `ABXInterest` objects from living forever.

To relieve users from the burden of having to manually update the focus representation (that is create and delete `ABXInterest` objects) every time the focus changes, we implemented an automatic mechanism based on earlier work by Bürger [18].

In Chapter 4, we described how an awareness system can observe the events of a user and use them to update this user's focus. After receiving an `ABXEvent`, the Relevance Assessment service first checks who is the originator of this event. It then does two things: one is to assess the relevance of that event for all users except for the originator (as described above); the other is to update the focus of the originator.

To actually update the focus of the originator, we implemented a rule based system, based on the design proposed by Bürger. The user now not only creates interests, but also creates rules that describe when and how new interests should be created (Figure 5.11).

Thus, when a new `ABXEvent` arrives in the Relevance Assessment service, it is first matched against all `ABXRules` of the originator. If it matches against

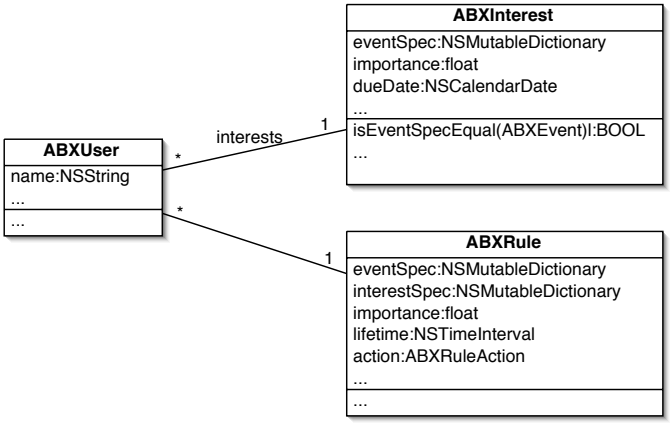

Figure 5.11: In addition the his `ABXInterest` objects, each user has a number of corresponding `ABXRule` objects as well.

the `ABXRule`'s `conditionSpec` attribute, a new `ABXInterest` is created according to the `interestSpec` attribute. The mechanism for matching an `ABXEvent` with the `conditionSpec` is identical to the mechanism for matching an `ABXEvent` with the `eventSpec` attribute of an `ABXInterest`.

The `interestSpec` attribute is different, however, as it is used as a template for creating new `ABXInterest` objects. While being a such a template, it does not fully specify the attributes that should be contained in the new `ABXInterest` object, but allows blank values. The Relevance Assessment fills in these blanks with attribute values from the arriving `ABXEvent`.

`ABXRules` also contain a flag attribute `action`, that can take the constants `DECREASE` or `INCREASE`. This takes care of the special case when a rule would cause the creation of an `ABXInterest` that is exactly the same as an already existing one. In that case the value of the `action` attribute is evaluated and the `importance` of the existing `ABXInterest` is changed to either the minimum or the maximum of the old and the new value.

Also an `ABXRule` contains an attribute `lifetime` that defines how long (in days) the rule remains active. Rules that have been longer around than indicated by their `lifetime` are automatically deleted.

To allow rule management (creation, modification, disposal), a functionality similar to the interest management described in the previous section has been implemented in the AWARENESS BUILDER client.

The window in Figure 5.10 can be split to show a list of `ABXRule` objects, a list of `ABXInterests` objects, or both (Figure 5.12).

To create new rules or modify existing ones, the user needs to invoke the Rule Inspector (Figure 5.13), that resembles closely the already described

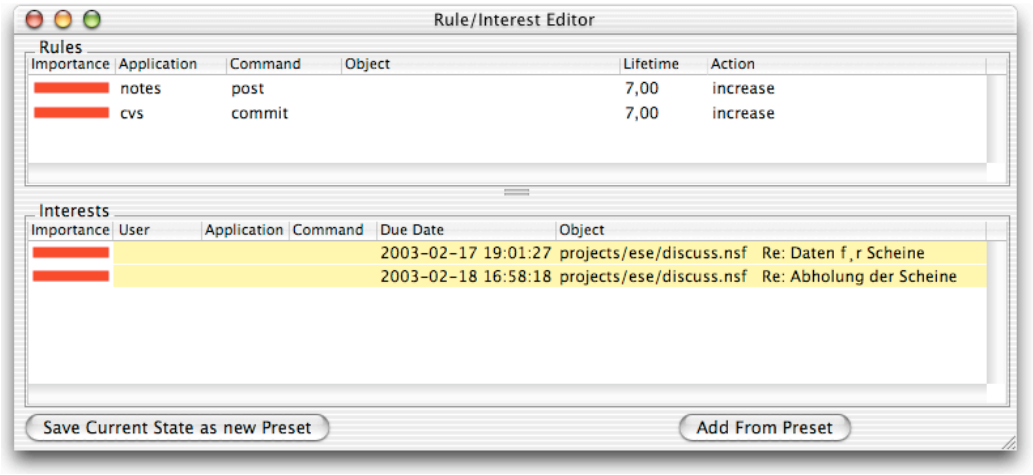

Figure 5.12: The *Rule/Interest Editor* window contains a split view that can be resized to show a list of `ABXRule` objects, a list of `ABXInterests` objects, or both.

Interest Inspector.

In the top part of the Rule Inspector window, the user can set the attributes of the `conditionSpec` attribute. The user can select an application, a command and an object. If he decides to leave an attribute blank, the resulting `conditionSpec` will match any value of the blank attribute.

In the example rule displayed in Figure 5.13 the application attribute is set to "notes" and the command attribute is set to "post", while the object field is left blank. This means that the rule acts every time the user posts something on one of the project's message boards, and creates an `ABXInterest` object according to the interestSpec.

In the bottom part of the Rule Inspector window, the user can set the attributes of the interestSpec attribute. Again, the user can specify attributes of the `ABXInterest` to be created. In addition, there is a checkbox next to each of the attributes. When checked, the attribute will take the value from the `ABXEvent` that triggered the rule.

In Figure 5.13, the box next to the object attribute is checked while the attributes user, application and command are left blank. This means that every time a user to whom this rule belongs posts something on a message board, a `ABXInterest` object will be created that will match all activities regarding the message (object) posted.

The second rule in Figure 5.12 acts on all "commit" commands in CVS: every time the user commits something to the CVS repository, a `ABXInterest` object is created that will match all activities regarding the file committed.

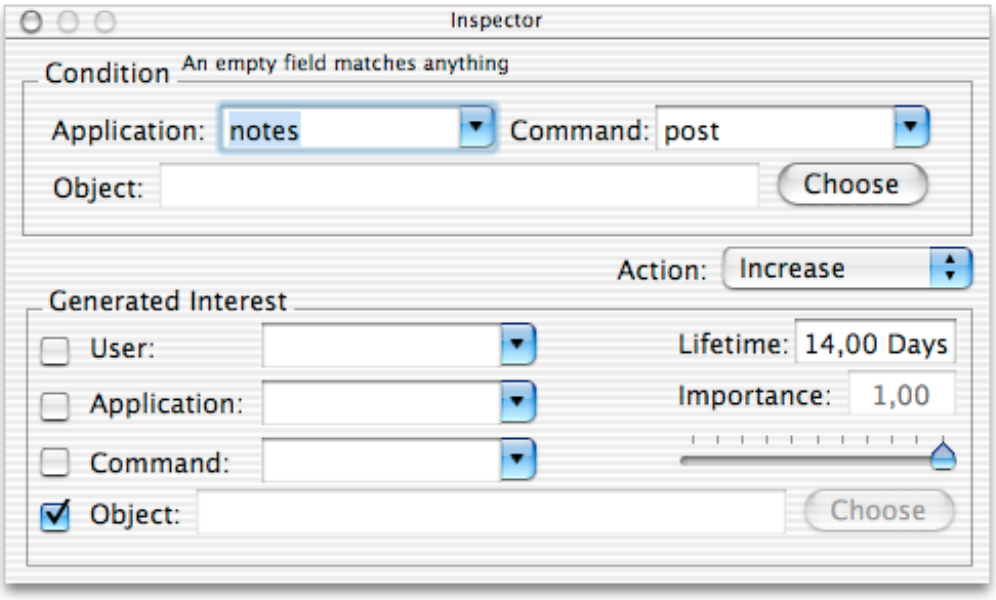

Figure 5.13: Invoking the Rule Inspector users can create new rules and modify existing ones.

Other attributes of the resulting `ABXInterest` include the `lifetime` (from which the due date attribute is calculated) and the `importance` value. The user can also choose whether he wants the `action` attribute to be set to `INCREASE` or `DECREASE`.

Now we can explain how the two automatically generated ABXInterest objects shown in Figure 5.9 and in Figure 5.12 came into existence. They were created by means of the first rule explained above: the user has obviously recently posted two messages on the *projects/ese/discuss.nsf* message board. The first message had the title "Re: Daten für Scheine" and the second message had the title "Re: Abholung der Scheine".

First, the `ABXEvents` corresponding to these actions were checked against the two rules defined for the originator. The first rule matched both of them as it was defined to match all post events within the LOTUS NOTES application. This meant that the template defined in that rule was used twice to create the ABXInterest objects that we see in Figure 5.9 and in Figure 5.12. The second rule did not match, and was ignored. After that the relevance of this `ABXEvent` for other users of the system was assessed as described in the previous section.

5.4 Dissemination Subsystem

After creating a `ABXNotification` object the Relevance Assessment service hands it over to the Dissemination service. The `ABXNotification` object is saved in a buffer associated with its receiver. Thus, any user is associated with a number of `ABXNotification` objects.

When the `ABXNotification` is added to the buffer, any AWARENESS BUILDER clients that are connected to the ABX server (a daemon called `abxd`) are immediately notified about a change in the buffer, so they can update their display accordingly.

The communication between the AWARENESS BUILDER clients and the ABXD server is currently based on the Distributed Objects contained in COCOA, so if other presentation devices — such as ambient displays — would need to be supported, than a proxy client would have to be implemented.

The `ABXNotification` objects are listed by the AWARENESS BUILDER client within a table view (Figure 5.14). The table can be sorted by column (the example shows the table being sorted by the time stamp attribute), and the user can rearrange the columns by dragging them with the mouse.

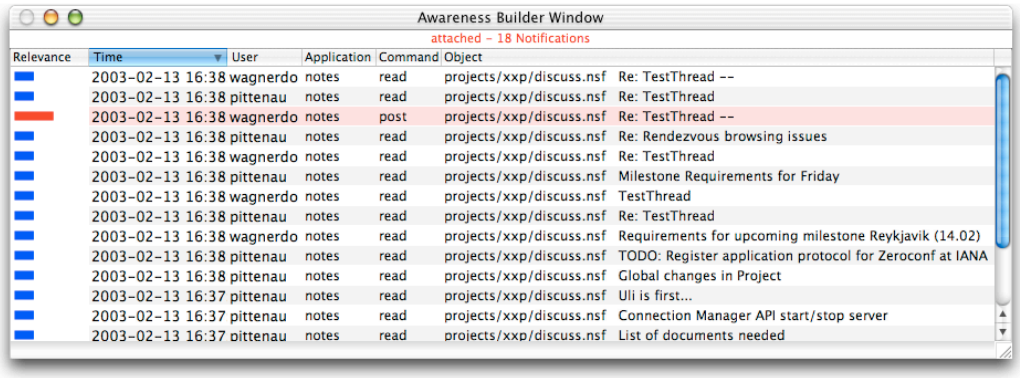

Figure 5.14: A user can view his ABXNotification objects in a table.

The example shows `ABXNotification` objects that result from a simple configuration using two `ABXInterest` objects: one `ABXInterest` object was defined for LOTUS NOTES "read" commands (using importance value 0.49 and a second object was defined for LOTUS NOTES "post" events (using importance value 1.0).

While we see a lot of events resulting from "read" commands, only one event resulting from a "post" command is highlighted. This highlighting occurs because the user has set his individual highlighting threshold to 0.5.

The client does not remove the ABXNotification objects from the server

side buffer unless they are deleted. So, a user without network access will not be able to view `ABXNotification` objects. We do not consider this to be a significant limitation, as the notification buffer is not meant as a storage facility for event logs (we will cover how we implemented such a facility for evaluation reasons in the next chapter). Rather, the user is supposed to be online to be able to receive important awareness event as soon as possible.

5.5 An Alternative Least-Effort Approach

Instead of building a separate Awareness System from scratch, it is certainly possible to add awareness features to each application (or use suitable existing mechanisms). Both applications currently supported by Collection Subsystem currently supports, CVS, and the custom bulletin board system based on LOTUS NOTES do provide limited awareness features. Some requirements for an Awareness System can be fulfilled using these features without having to build a system like AWARENESS BUILDER. In this section, we discuss the features that can be used to provide awareness. Also, we will compare this alternative approach to our approach.

The Concurrent Versions System CVS supports invoking scripts after certain operations (e.g. "commit") are executed. In fact, we use this mechanism to implement our awareness sensor for CVS in AWARENESS BUILDER. This mechanism is quite often used to send "commit" emails to individual developers or developer teams. Using this method, team members can acquire awareness about a certain type of events through the familiar email system.

It is considered good practice (e.g. Maguire [50]) to configure CVS to send these emails to the team lead and the members of ones own team. Otherwise, the combined number of such emails might become to big on large projects and make the recipients feel dissatisfied [38]. Also, some users might not want to have their mailbox polluted with automatically generated status emails at all.

Other than being very tightly tied to a single delivery method (email), this method requires the CVS administrator to maintain the groups of users who are supposed to get certain notification emails. This requirement puts the burden of both manually maintaining the technical configuration and of deciding who is supposed to get the notifications on a human.

With this method, the collection is an implementation detail of CVS, relevance assessment is left to the human administrator, and dissemination uses an existing email system. While the initial implementation is quite simple, because almost no custom code (other than a simple "commit" script) is required, the precision of the notifications is either low (whole team re-

ceives simple "commit" emails), or the ongoing maintenance costs are high (administrator constantly monitors users interests and maintains a complex "commit" script). Also, the end user has no way to change the configuration of the system all by himself.

The LOTUS NOTES bulletin board system also provides notification features that can be used to acquire awareness. Similar to the above method available in CVS, the features in NOTES rely on an external email system for notification delivery. However, the configuration of the notification features does not require an administrator, and can be done by the end user all by himself.

The end user can specify can configure each bulletin board to send email notifications whenever a posting was created by a certain author, belongs to a certain category and/or contains a certain word or a phrase.

With this method, the collection is again an implementation detail of the application, that is LOTUS NOTES. Relevance assessment is automated — however, there is no notion of relevance and the user is required to manually add, change and remove rules to keep the stored interest profile in sync with his or hers real interest profile.

The implementation is even simpler that for the alternative CVS method, because no administrator is needed. Still precision is either low, or maintenance effort for the individual user is high. In the past, we observed that only the most sophisticated users would use this mechanism and even those users would only very rarely change the rules.

Both methods combined to provide limited awareness support that is similar to the effect of using AWARENESS BUILDER. However, in both methods, support for Relevance Assessment is very limited and relies heavily on human configuration. Also, Dissemination relies on an external email system.

5.6 Summary

In this chapter, we have covered the architecture, the design and implementation decisions we had to make to implement a working awareness system for distributed projects: AWARENESS BUILDER.

We have discussed in detail the lifecycle of an awareness event from the point when it enters our system till the point when it is presented to the user after being assigned an importance weight.

In the next chapter, we will cover additions to the system that were needed for evaluation, as well as the experiments we have conducted using our system.

Chapter 6

Evaluation

In the previous Chapter, we have described a proof-of-concept implementation of the AWARENESS BUILDER system, based on the requirements for awareness systems outlined in Chapter 2 and Chapter 3. In this chapter, we present the results of several experiments we conducted using this prototype implementation.

We first look at evaluation methods that can be used to assess the effectiveness of an awareness system in a distributed development setting and present both our method and our evaluation testbed for quantitative evaluation (Section 6.1). We then review the results of some experiments that were conducted to calibrate and assess the effectiveness of AWARENESS BUILDER.

These experiments fall into two categories: a retrospective set of experiments using historical data from past distributed development project courses to calibrate the awareness system and to improve the overall usability and usefullness of the system (Section 6.2), and a prospective set of experiments during an an ongoing distributed project course (ARENA). While the first study was limited by the amount and the quality of the available historical log files, the second study benefited from customized logging and from the availability of the project participants for questioning. Thus, the real impact of the system could be observed (Section 6.3).

The chapter ends with a short conclusion that summarizes our main experimental results (Section 6.4).

6.1 Methods

In a survey of more than 400 research articles, Tichy et al. [79] have found that 40% of all surveyed computer science publications that had required experimental validation, had no such validation at all. For software engineering

publications, the number was more than 50%. The numbers in the unrelated research fields chosen for comparison, Optical Engineering and Neural Computation, were only 15% and 12%, respectively.

As a response, many authors have advocated more experimentation in computer science and in software engineering (e.g. [78]). Sneltig suggests [75] that all ideas about tools and processes should be at least implemented and the resulting proof-of-concept implementations should become publicly accessible, to allow for later evaluation. However, the evaluations phase must not be omitted.

Empirical studies are key to understand the costs and benefits of software tools such as AWARENESS BUILDER. Perry et al. [59] present a roadmap for performing empirical studies in software engineering that we can follow to describe possible ways to evaluate the usefulness of our system.

One component of this roadmap is to formulate a good hypothesis or question to test. While the very important questions about the benefits of our tool are easy to state (e.g. "Does AWARENESS BUILDER improve the output of a project?"), they are hard to answer if there is only a small number of studies. Thus, Perry et al. suggest that the most important question might be not the most insightful one.

Also, easy to state questions like "Does AWARENESS BUILDER improve the output of a project?" are very abstract and need to be refined into more low-level, concrete questions, such as "Does AWARENESS BUILDER shorten the schedule of the development project?" or "Does AWARENESS BUILDER improve the quality of the artifacts created?" Even these more concrete questions are too hard to answer without a large number of studies.

Another component is the design of the study in terms of dependent and independent variables. This is an inherently hard problem in software development as developers usually do not build copies of the same system. Thus, a controlled environment that allows to measure the impact of tools and methods is hard to establish.

Perry et al. suggest to choose short, frequent and inexpensive tasks such as bug-fixing and testing instead. They also suggest to use historical data instead of live measurements of subjects as they perform predetermined tasks. Version control systems (e.g. CVS) not only allow to recreate the state of a software system at an arbitrary point in time, but also contain a lot of additional information such as what part of the system was changed, when it was changed and who changed that part.

After performing the experiments, the resulting data needs to be subjected to either quantitative or qualitative analysis. Quantitative analysis describes and compares numeric data. Qualitative analysis deals with non-quantified data that typically has been obtained through questionnaires, in-

terviews and observations.

6.1.1 Questions

We decided to focus our evaluation on two qualities: the quality of Relevance Assessment service and on the second being the usability of the system.

To assess the usability of our system, we established the following questions:

- How often do users use our system?
- How do they like the user interface?
- How do they like the interest/rules mechanism?
- How useful do they find our system for their work in the project?
- How often does a notification trigger a communication beween users?
- Which activities are being monitored — those of the members of a user's own team, members of other teams or both?
- Which types of events are monitored — NOTES, CVS or both?
- Does our system prevent CVS conflicts?

In addition, we identified two questions regarding the quality of the Relevance Assessment service:

- Does the number of awareness events in a typical project justify the effort to design, implement and use the Relevance Assessment service?
- What is the relevance distribution calculated by the Relevance Assessment service?

To answer these questions, a case study combined with standard usability engineering techniques such as observations, questionnaires and interviews [52] was used. . We also used descriptive quantitative analysis to compare data obtained from the prospective experiment to data from the retrospective experiments. We cover our case study, the deployment in the ARENA project in Section 6.3.

6.1.2 Qualitative vs. Quantitative Evaluation

Many publications on awareness systems either lack evaluation results or base their evaluation results on qualitative methods only. Very often, the authors describe their own experience using their own systems or quote informal conversations with pilot users.

For example, Dourish and Bly [26] describe their method to evaluate PORTHOLES after some initial *informal and anecdotal* (sic!) observations have been made:

> In order to get more detailed feedback on the use of, and reactions to, our prototype Portholes system, we asked a group of fifteen users to note their usage of Portholes over a three-day period and to fill out an electronic questionnaire. The questionnaire also asked open-ended questions regarding features they liked and disliked.
>
> We received eleven responses by electronic mail. While we do not believe we're ready to "quantify" the effects of awareness, we can observe some patterns in the typical use of Portholes.

A solely qualitative approach is typical for early, pioneering work on awareness systems. When Bly et al. [11] describe the impact of their MEDIA SPACES system, they state:

> These particular examples have emerged from our own experiences, from videotaped data, from internal reports, and from anecdotal evidence.

Publications often focus on the technology used to present awareness information to the user only. A typical example is the description of the AMBIENTROOM proof-of-concept prototype by Wisenski et al. [83].

Few publications present quantitative data. While Prinz et al. [67] base their results on informal user feedback, they do provide at least some quantitative data on the workspace environment they seek to improve with their TOWER system.

With statistics based on log files for a period of approximately 200 days of a shared workspace, they derive evidence "that users actually meet in a shared virtual project or workspace (...)".

In our evaluation, we conduct a case study over a period of one semester (approximately three months) with 18 users. Our goal ist to answer the questions about usability stated in the previous section. To achieve this goal, users are required to fill our a web based questionnaire after the case

study and rate their experience with AwarenessBuilder. The results are summarized later in this chapter in Section 6.3.

To answer the questions about the quality of the Relevance Assessment service, we use a quantitative method and extend AwarenessBuilder with an experimental testbed. Our testbed consists of several extensions to awareness builder — some built into the tool itself, others being external to it. In the following sections, we describe our testbed and the way we obtained suitable input data from historical projects. Later, we describe the data we obtained using this testbed on four different projects.

6.1.3 A Quantitative Evaluation Method

In this section, we present a quantitative evaluation method which allows to observe the flow of information that an awareness system creates during the course of a project. It also allows to observe the flow of information that an awareness system *would have* created on past projects that did not use an awareness system. In addition, repeated application of the method on the same project allows to fine-tune an awareness system.

First, we remember that any awareness system constantly collects events and, for each user, assigns an importance weight to each of the collected events. It then disseminates interesting events to its users sending notifications that contain the respective addressee, the event and the assigned importance weight.

Thus, if we consider E to be the set of all awareness events that can be created by a given environment and N to be the set of all awareness notifications that a given awareness system can create, than an awareness system, over time, simply computes the function $f : \mathcal{P}E \rightarrow \mathcal{P}N$ (where $\mathcal{P}E$ denotes the powerset of E and $\mathcal{P}N$ denotes the powerset of N).

Assuming that the configuration of the awareness system remains static over time (no parameters are changed while the system runs), for any project p, we can calculate the set of notifications for the project N_p if we obtain the set of events E_p that were created during the project.

For certain classes of projects, including our application domain — software development — obtaining the set of events E_p is possible, even for archived projects. Software configuration management and communication systems preserve most the information required for this task.

Once the set of events E_p is available, the awareness system can be used to compute the set of notifcations N_p, again and again. As long as the parameters of the awareness system are not changed, the computation results will remain the same, no matter how often the experiment is repeated.

If we can obtain multiple sets of events (e.g. from multiple projects), the method allows us to observe how changes in the input data influence the output data and to look for common characteristics.

Also, focusing on one particular project and applying the method repeatedly while changing certain parameters of the system allows us to observe how changes of the parameters influence the output data in terms of characteristics like volume and distribution.

We can summarize our method as five consecutive steps:

1. Obtain a set of events E_p
2. Define the parameters of the awareness systems
3. Calculate the set of notifications N_p
4. Repeat with a different set of events or with different parameters
5. Perform Analysis

This method does not make any assumptions about the awareness system used. In practice, small changes and extensions are required which transform an awareness system into an evaluation testbed. The issues are presented in the next section.

6.1.4 Evaluation Testbed

To support our quantitative evaluation method presented, two logging mechanisms are needed: one for notifications and one for events. The logging mechanism for notifications creates a Notification Log containing all notifications created by ABXD, the AWARENESS BUILDER server, during the course of a project. After the project is finished, this Notification Log can be subjected to a detailed post-mortem (off-line) analysis.

The functionality required to create such a log was built directly into ABXD, the AWARENESS BUILDER server, and located in the Dissemination service. The Notification Log contains an entry for each notification created. Each entry consists of the attributes of the corresponding ABXNotification: time stamp, originator name, application name, user action, object descriptor, importance value and the name of the receiver.

The event logging mechanism creates an Event Log containing all the events received by ABXD, during the course of a project. The Event Log contains an entry for each event received and each entry consists of the attributes of the corresponding ABXEvent: time stamp, originator name, application name, user name and object description.

After the project is finished, the Event Log can be analyzed on its own, or can be compared to the Notification Log to see how the system weigthed the events received for the individual users.

The Event Log allows to add *limited repeatability* to the evaluation. While the project itself cannot be repeated, it is possible to repeat or "replay" the Event Log again and again — after changing the system configuration in some way (e.g. defining new rules or changing the way relevance is calculated) — and analyze the impact it has on the Notification Log. To enable limited repeatability, we have implemented a separate tool, the LOG PLAYER, that allows to "replay" a stored Event Log.

The replay speed can be adjusted individually using a slider — the speed range goes from real-time (meant for debugging) to as fast as possible (meant for log generation for evaluation purposes). The progress can be observed in a status area where the number of events already sent, the total number of events and the time stamp of the most recently sent event are displayed.

Implementing LOG PLAYER as an external application (rather as a part of the server) reduces the complexity of ABXD. In terms of a simple LOC (lines of code) metric, the LOG PLAYER functionality would add almost 10% to the size of ABXD (considering only ABXD-specific code, and not taking into the account the AWARENESS BUILDER Framework code, that is shared by both the client and the server).

Moreover, while ABXD, being a server, was implemented as a Unix daemon, the LOG PLAYER includes a graphical user interface to simplify usage. Integrating ABXD and LOG PLAYER in one program would either require to abandon the GUI for the LOG PLAYER functionality, or to integrate the LOG PLAYER functionality in the AWARENESS BUILDER client.

Having LOG PLAYER as a separate application allows to use it as a test driver for ABXD. It was designed in such a way that there are no changes on the server (ABXD) side, and it is just using the standard event collection service of the server. Rather than writing a special-purpose test driver application that would generate artificial events and send them to the server, the LOG PLAYER can supply real, historical events to the server.

The drawback of this solution is that the notion of time of the LOG PLAYER and the notion of time of ABXD can be different when the user chooses to accelerate the replay speed. This is a result of a trade-off between replay accuracy and ease of integration. To allow easy implementation of application sensors and to solve clock synchronization issues in real projects, we decided to time stamp events in the server, rather than requiring each application to provide time information. We accepted the resulting loss of replay accuracy and will point out the implication when we discuss our retrospective experiments.

So, the combination of the AWARENESS BUILDER server ABXD with the extensions mentioned above and the AWARENESS BUILDER LOG PLAYER constitute a testbed for both the evaluation of the effectiveness of the AWARENESS BUILDER server's relevance assessment service as well as testing the robustness and scalability of the server.

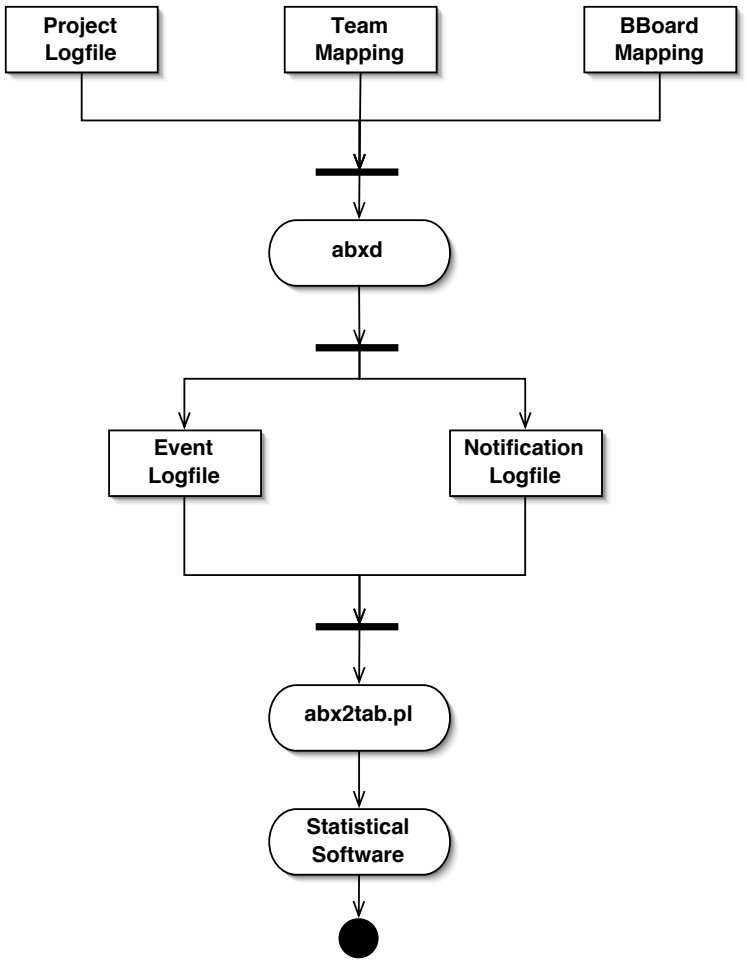

Figure 6.1: The testbed can be used to evaluate the ABX with different inputs and different configurations again and again.

Figure 6.1 illustrates the evaluation process: first, an event log file has to be supplied together with the usual ABXD project configuration files containing the people/team mapping and the known message (internally known as BBoards) boards. Also, the server needs to be configured with a set of presets containing an initial set of rules — in a real project, this initial set of rules can (and usually will be) modified by the users, while in a playback

(simulated) project, it remains static.

Then, LOG PLAYER is used to playback the Event Log file and the ABXD is configured to create Notification Log file. The creation of another Event Log file is optional. The resulting files can be processed by a script that converts them to a format suitable for easy analysis, using e.g. MICROSOFT EXCEL.

6.1.5 Log Reconstruction

We reconstructed an event log for three projects (PAID, STARS, and TRAMP) including the following classes of events:

- LOTUS NOTES "post" Event: a user has posted a new message on one of the project's Lotus Notes message boards
- LOTUS NOTES "read" Event: a user has read a message on one of the project's Lotus Notes message boards
- CVS "commit" Event: a user has commited (checked-in) a file to the project's version control system CVS

These projects did not use AWARENESS BUILDER, so we do not know if the users would have changed any of the initial rules and interests defined for them during the initial setup of the relevance assessment service during the project. In our analysis, we therefore assume that the users just worked with initial setup of the system without changing it.

To reconstruct the Event Log we used a multi-step process using various tools and scripts described in Figure 6.2.

First, two separate Event Logs for the two supported applications, LOTUS NOTES and CVS were created. As LOTUS NOTES uses different user names than CVS (full names rather then Unix-style login names), we had to map the user names to a common user identifier before we could merge the resulting Event Log files. Our system uses Unix-style login names as user identifiers, so we did not need to do any mapping for the CVS Event Log, but only for the LOTUS NOTES Event Log.

The resulting Event Log was used directly as an input for the LOG PLAYER application described in the previous section.

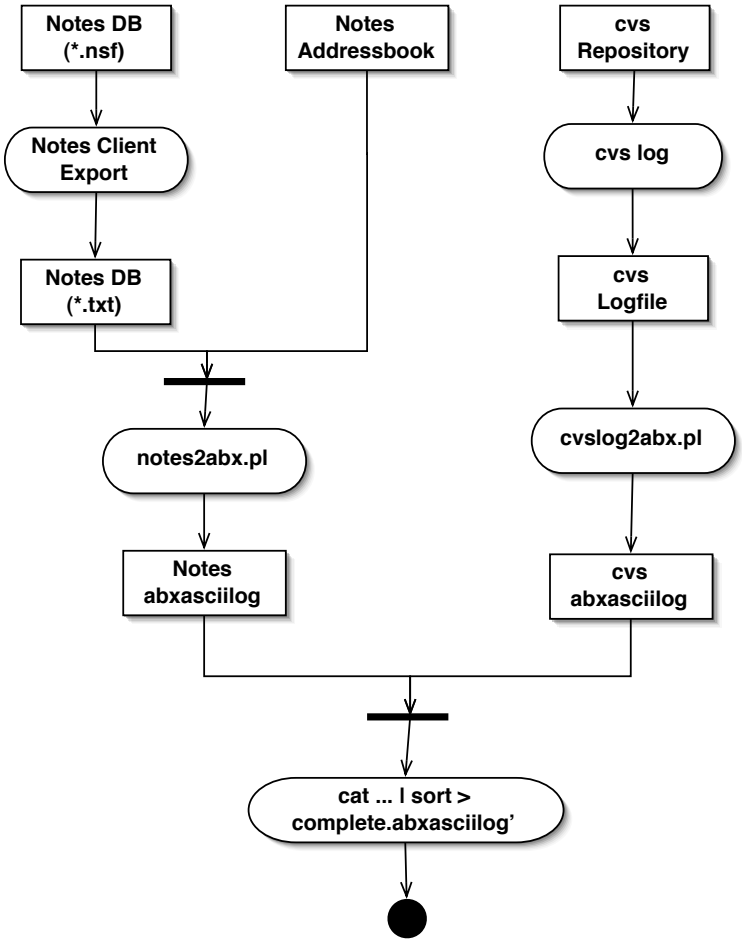

Figure 6.2: Log file reconstruction is a multi-step process.

6.2 Retrospective Experiments

6.2.1 Common Setup

For each project (PAID, STARS and TRAMP), we set up a separate AWARENESS BUILDER installation. We configured the AWARENESS BUILDER server, ABXD, using project specific people/team mapping files and files describing the message boards used. Then, we provided an initial rule/interest setup file.

For the purpose of our experiments, the initial setup consisted of two rules to track actions of others that are related to the current focus observer. To keep track of that focus, we have defined the following two rules, using the automatic focus update mechanism described in Chapter 5:

- every time a user commits a file to the CVS repository, an interest for all actions on this file is created,
- every time a user posts a message on one of the project message boards, an interest for all actions on this message is created.

Both rules were instantiated for every user known to be a participant of the project and created interests with an importance weight of 1. The resulting interests had the lifetime set to one week. However, due to our decision to time stamp the events on the server, in the simulation the interests would exist for the lifetime of the (simulated) project, because we did not simulate the projects in real-time but rather accelerated them to the maximum.

While our rules would only generate interests for objects that the users would modify, the users would be also delivered notifications about other, closely related objects. This property of our system is due to the model mechanism described also in Chapter 5, that encapsulates the notion of event proximity.

So, for example, users did not only get notifications about files that they have committed, but also about files that include their file, files that are included by their file, of files that are in the same directory as their file. Similarly, they would also not only get notifications about the message they posted, but also about replies posted to that message.

6.2.2 Variables

The random variable *project* takes one of the values PAID Event Log, STARS Event Log or TRAMP Event Log. For our retrospective evaluation, we decided to include two additional evaluation variables.

The second variable *relevance formula* is used to calculate the individual relevance values for awareness notifications that are delivered to the users of the system. The values of this variable are the particular formulas described in detail in Chapter 4. The following formulas are variants of those formulas, which employ a proximity factor $proximity()$ that takes object and event proximity into account, as described in Section 5.3.2:

Formula 4.1: a simple yes-no formula for nonoverlapping interests,

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ 1 & : \quad e \in R_u \end{cases}$$

Formula 4.2: a simple formula for overlapping interests,

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ \frac{|\{I_{1 \leq i \leq n} \subseteq R_u : e \in I_i\}|}{n} & : \quad e \in R_u \end{cases}$$

Formula 4.3: similar to Formula 4.1, for weighted interests,

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ importance(I_i) \cdot proximity(I_i, e) & : \quad e \in I_i \subseteq R_u \end{cases}$$

Formula 4.4: the first formula for weighted, overlapping interests that uses the average function to combine several importance values into one relevance value,

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ \frac{\sum_{i=1}^{m} importance(I_i) \cdot proximity(I_i, e)}{m} & : \quad e \in I_i \subseteq R_u \end{cases}$$

Formula 4.5: the second formula for weighted, overlapping interests, but one that uses the maximum function instead of the average function,

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ \max(importance(I_i) \cdot proximity(I_i, e)) & : \quad e \in I_i \subseteq R_u \end{cases}$$

Formula 4.6: the third formula for weighted, overlapping interests. This one uses the sum function and thus allows the distinct importance values to cumulate.

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ \min(\sum_{i=1}^{m} importance(I_i) \cdot proximity(I_i, e), 1) & : \quad e \in I_i \subseteq R_u \end{cases}$$

These are the general formulas that our system supports. However, during the retrospective experiments, only a limited set of interests is fed to the system. Thus, only interests with a importance value of 1.0 exist in the system and any point in time (see previous section). This simplifies the formulas significantly, as $importance(I) = 1.0$ for any interest I.

As a result of this simplification, the formulas 4.3 – 4.6 can be restated (for the retrospective experiments only) as:

Formula 4.3:

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ proximity(I_i, e) & : \quad e \in I_i \subseteq R_u \end{cases}$$

Formula 4.4:

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ \frac{\sum_{i=1}^{m} proximity(I_i, e)}{m} & : \quad e \in I_i \subseteq R_u \end{cases}$$

Formula 4.5:

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ \max(proximity(I_i, e)) & : \quad e \in I_i \subseteq R_u \end{cases}$$

Formula 4.6:

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ \min(\sum_{i=1}^{m} proximity(I_i, e), 1) & : \quad e \in I_i \subseteq R_u \end{cases}$$

The third and final evaluation variable are the *model weight factors* i and f used in the `ABXCVSModel` to combine the results of the (logical) inclusion hierarchy metric (I) and the (physical) filesystem hierarchy metric (F) to calculate the object proximity $p_{object} = i \cdot I + f \cdot F$ (for details refer to Section 5.3.2) between source code files. Possible value include:

(i = 0.5, f = 0.5) averages the effect of both metrics,

(i = 0.1, f = 0.9) assignes more weight to the results of the file hierarchy metric,

(i = 0.9, f = 0.1) assignes more weight to the results of the inclusion hierarchy metric.

6.2.3 Event Log Processing

Given the three evaluation variables *project*, *relevance formula* and *model weight factors* each with three, seven and three, respectively, possible values, we ended up with a total of $3 \cdot 7 \cdot 3 = 63$ evaluation runs.

To perform these runs, six computers where used over a course of about two months. We used Apple PowerMacintosh G4 Dual 1,42 GHz computers equipped with 1 GB RAM. The average duration for one evaluation run at maximum speed was 39 hours and 12 minutes for PAID, 9 hours and 6 minutes for TRAMP and 13 hours and 7 minutes for STARS.

Each evaluation run resulted in a file `notifications.tab` that contained the Notifications Log for the given set of parameters. For easier evaluation, each result has been stored in a separate directory (Figure 6.3) together with the corresponding Event Log (`events.tab`) file.

Thus, each evaluation run corresponds to computing the function:

$$notification.tab = ABX(project, relevanceformula, modelweightfactors)$$

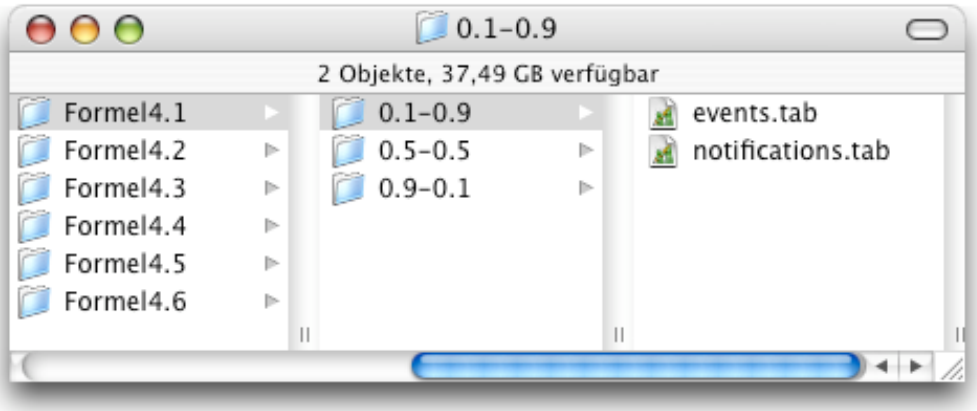

Figure 6.3: Each evaluation run resulted in a pair of files that contained the respective Event Log and Notifications Log for the given set of variable values.

6.2.4 Event Space Characterization

A first look at the different projects we had historical data for from a high-level perspective reveals (Table 6.1) significant differences between the projects.

The projects differ in both the number of events and the number of notification that would have been triggered by those events.

For a given project, the number of events remains constant by definition of our evaluation method, no matter which *relevance formula* or *model weight factors* we use. It is somewhat surprising, that the number of notifications remains constant as well, e.g. 69.418 notifications in TRAMP, independent of the formula used.

In ABX, a notification is triggered when the importance value calculated by the Relevance Assessment service exceeds the global threshold, which in our case is set to 0. Obviously, our parameters (as we will later see) influence the relevance values of the notifications created by our system, but do not influence the number of calculations that result in values greater than 0.

For the purpose of analyzing the work done by the Relevance Assessment service, each project can characterized by three values: the number of participants, the number of events that have been received by ABX and the number of events that were created as a result of receiving these events using the initial set of rules described in the previous section.

To put this data into perspective, we have also added to the number of lines of code and the number of code files in the repository at the end of the project for each project. These numbers provide a simple metric for the

	PAID	STARS	TRAMP
Logged Participants[1]	110	**121**	78
Final Lines of Code	**231.601**	58.435	25.071
Final Code Files	**1.673**	611	269
Events	**85.948**	60.136	58.223
Notifications	**92.919**	54.691	69.418
Events*Participants	**9.458.240**	7.276.456	4.541.394
Events/Participants	**782**	497	746
Notifications/Participants	845	451	**890**
Filtering Ratio $(1 - \frac{Notifications}{Events*Participants})$	**99,0%**	99,2%	98,5%
Notifications/Events	1,1	0,9	**1,2**

Table 6.1: Input and Output Data Volume (bold face denotes maximum values for each row).

overall size of the systems developed in our projects.

The PAID system was by far the largest system developed in the sample (even though the STARS project had more participants). Thus, it's hardly surprising that the most events would have been received during the PAID project. During the course of that project, 110 participants have generated 85.984 awareness events that have triggered 92.919 notifications.

The numbers show that the Relevance Assessment service that we have proposed for awareness system is necessary: without relevance assessment and the resulting filtering of events, each participant would have received a notification for all events of project, that is 85.984 notifications. A total number of 9.458.240 notifications have been distributed to all project participants and thus massively contribute to the information overload problem.

Using our Relevance Assessment, only less than one hundredth, namely 92.919 notifications were created. That's only 845 notifications per participant (assuming an even distribution) — a number that users can be expected to be able to cope with during the course of a project that lasts several months.

The numbers for the other projects (STARS and TRAMP) support this observation. The total number of notifications without a Relevance Assessment service is in the millions, while the actual number of notifications is two orders of magnitude less.

The actual filtering ratio (the percentage of potential notifications that are not created when the Relevance Assessment service is active) varies slightly across the projects. It is higher — 99,2% — when the awareness events have on the average triggered fewer notifications (e.g. 0,9 in STARS)

and lower — 98,5% — when they have triggered more notifications (e.g. 1,2 in TRAMP).

One possible conclusion from the lower filtering ratio — and the high number of notifications per participant (the highest in our sample) — in the TRAMP project is that the level of collaboration was highest in TRAMP. Anecdotal evidence resulting from private communication with project participants supports this conclusion.

However, the data presented in this section supports our claim that the Relevance Assessment service *reduces the number of notifications significantly.* In fact, it reduces that number by two orders of magnitude, filtering out the "unimportant"[2] events.

6.2.5 Event Types

In this section, we take a closer look on the distribution of events across the different event types. In the experiment, we chose to work with three distinct event types: CVS commit events, LOTUS NOTES post events, and LOTUS NOTES post events. Again, the distribution of event *types* is neither influenced by the *relevance formula*, nor by the *model weight factors*, but by the *project* only.

Table 6.2 summarizes the occurrences of each of these three event types in the project event and notification logs. With occurrence in the range of 81,9–91,4%, the LOTUS NOTES read events are — not surprisingly — by far the most frequent events in our projects.

The projects were distributed in both time and space. Thus, the participants have to rely on asynchronous media for communication. While the participants have access to individual email accounts, they are encouraged to use our LOTUS NOTES based message board system for all project related asynchronous communication. The large number of read events provides evidence that the messages posted on the message boards are considered important and are being read by most participants.

While the LOTUS NOTES post events are in a close range of 7,6–9,8%, the CVS commit events are spread between 1,0% (in TRAMP) and 8,3% (in PAID). Obviously, the software configuration management process differed significantly between the evaluated projects.

We did not look at the granularity of the individual commits, therefore we cannot tell for sure whether this spread is due to a focus on implementation in certain projects (i.e. PAID and STARS) and a focus on analysis and design

[2]There is no way to tell if the system filtered out the right events at this point, of course.

		CVS commit			post	LOTUS NOTES read	
		Frac.	No.	Frac.	No.	Frac.	No.
PAID	Events	8,3%	7.171	9,8%	8.409	81,9%	70.404
	Notifications	7,0%	6.505	3,6%	3.312	89,4%	83.102
STARS	Events	7,0%	4.181	8,4%	5.065	84,6%	50.890
	Notifications	12,1%	6.604	2,7%	1.493	85,2%	46.594
TRAMP	Events	1,0%	580	7,6%	4.413	91,4%	53.230
	Notifications	5,4%	3.720	2,4%	1.693	92,2%	64.005

Table 6.2: LOTUS NOTES read events are the most frequent event type.

in others (i.e. TRAMP). Table 6.1 shows that the code volume produced during the TRAMP project was below average.

However, the picture becomes clearer if we take a look at the events that actually did trigger a notification. In Table 6.2, we have also included the numbers (and fractions) of notifications that were triggered by a certain event type. With these data, we can — for each event type — compare the number of events and the number of notifications in each project.

LOTUS NOTES read events tend to trigger a similar number of notifications in each project. This comes not surprisingly, because in our experimental setup, the writer of a message is always notified about read events related to his message.

So reading a message triggers at least one notification to the author of that message and may also trigger more notifications if the message is nested in a thread (this explains why in some projects there are more notifications about read events than there are read events).

Here, the STARS project is an outlier: there are more read events than there are notifications about read events. This can be explained when we consider that reading ones own messages does not trigger a notifications. One possible interpretation is that the participants of the STARS project used to read their own messages more frequently than their counterparts in the other projects.

With these findings we could be concerned about users being overwhelmed with read events. To limit the number of read notifications, one could simply reduce the lifetime of interests created for one's own postings or create two rules instead of just one for one's own messages: one rule that would create high importance and long lifetime interests for post events and one rule that would create lower importance and shorter lifetime interests for read events.

When we move to the LOTUS NOTES post events, we see that there are

consistently less notifications about post events than there were post events. This, too, can be explained with our experimental setup. Without manual configuration, the rules of the initial setup create interests for message that are related to one's own messages: that is, messages that are answers to one's message. This means that notifications are only created for message postings that are answers to some other message and that no notifications are created for message posting that start a new discussion thread.

The data shows that in the surveyed projects, much less than 50% of all messages are replies to other messages, and thus only a fraction of all message postings trigger a notification with the default set of rules. If we assume that users tend to be quite interested in new messages, this result might indicate that the initial setup is not sufficient, and should include an default interest for certain postings, such as those on one's own team message board.

Finally, when we compare the number of CVS commit events to the number of notifications that were triggered by such events, the resulting ratios are different for each of the projects. In the PAID project, they are slightly less commit notifications than there are commit events, while in the STARS project it's the other way around: there are slightly more commit notifications than there are commit events.

One possible explanation is that while in PAID the responsibilities for source files were more clearly divided between the project's participants, in STARS there was more overlap. When the responsibilities would have been divided such that every participant worked on a separate set of files (without any overlap), then we could have observed no commit notifications at all — this is a consequence from our choice of the initial set of rules.

It appears that there was even more overlap in TRAMP, where there would have been more than six times more commit notifications than commit events. Another explanation is that the granularities of the commits were higher.

If the commit granularity was similar to the other projects, then it is possible that the developers in TRAMP did not have clear cut responsibilities for certain files, but where rather working cooperatively on the implementation. We expect that this is the scenario, where the developers will profit the most from improved awareness support, as conflicts can easily occur and the progress of others has direct impact on ones own work.

6.2.6 Importance Distribution

This section discusses the importance distribution of the notifications created. The importance distribution is a table which, for each possible importance value, indicates the number of Awareness Notifications that were

assigned this importance value. As the number of distinct importance values is very large (the attribute `importance` is implemented as a floating point number), we use intervals of length 0.1 when presenting the data.

First, we observe how different relevance formulas and model weight factors impact the importance distribution of notifications in each of our sample projects. Based on these observations, we then select one formula and one model weight factor pair (parameters that we also use later for the prospective experiment with the ARENA project) to compare the individual projects' importance distributions.

Formula 4.1

When Formula 4.1

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ 1 & : \quad e \in R_u \end{cases}$$

is used, potential notifications are assigned either zero or one. As all values that are equal to zero are discarded by our system regardless of the formula used, all notifications that are actually delivered to users have an importance value of 1.0.

Also, in this case, the model weight factors do not make any difference. If the proximity calculated for a event/interest pair is not inifinite (nil in our implementation), the system will assign 1.0 no matter which weight factors have been chosen.

Thus, this formula does not seem to be reasonable choice for our system.

Formula 4.2

When Formula 4.2

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ \frac{|\{I_{1 \leq i \leq n} \subseteq R_u : e \in I_i\}|}{n} & : \quad e \in R_u \end{cases}$$

is used, the importance value distribution looks more reasonable (Figure 6.4). The flaw of this formula lies in the fact that importance values are diminished with the number of non-overlapping interests created for a particular user.

Consequently, on one extreme end, only users with just a few (overlapping) interests can receive a notification with an importance value 1.0. For example, in TRAMP, where a total of 53 notifications with an importance value of 1.0 where delivered, all of those notification where targeted to just

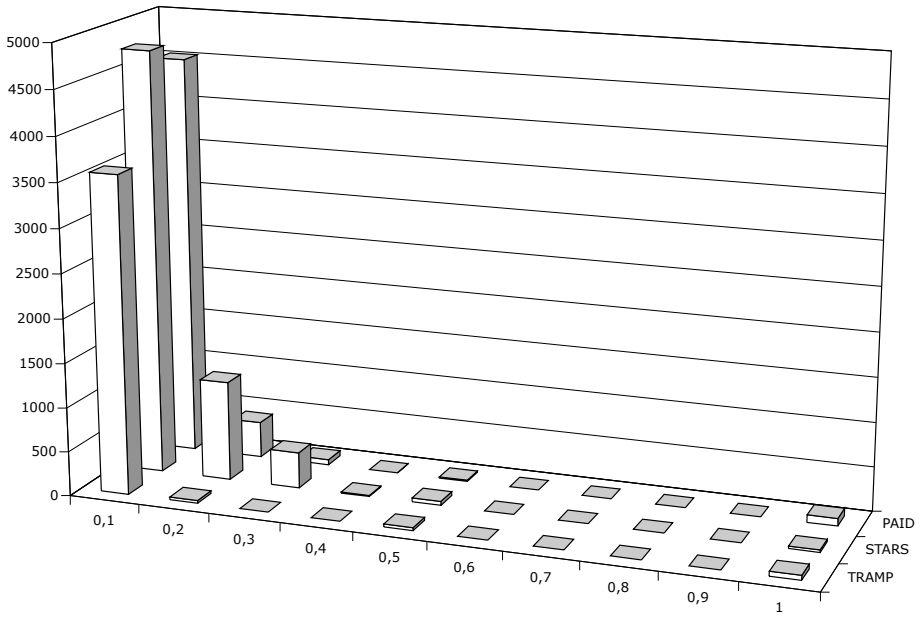

Figure 6.4: The distribution of importance values in notifications regarding CVS commit commands for Formula 4.2.

one particular user. Through the duration of the whole project, this particular user has contributed only two modifications to the CVS source tree (and no postings to the message boards) and thus had two interests created for him in the system. These two interests also overlapped, because both files resided in a common directory.

Obviously, this formula does not work well in our environment, because most users have overlapping and non-overlapping interests. E.g. a user with just one posting and one contribution to the CVS tree would have two non-overlapping interests. So our environment lies at the other extreme end with lots of overlapping and non-overlapping interests that result in small (less than 0.1) importance values.

Also, this formula — like Formula 4.1 — does not take the proximity factor into account. What counts here is just whether the proximity computed by the Model-Subsystem is infinite (negative interest match) or not (positive interest match).

	$(i = 0.1, f = 0.9)$	$(i = 0.5, f = 0.5)$	$(i = 0.9, f = 0.1)$
0.10	6048	0	0
0.20	0	6048	0
0.30	0	0	6048
0.33	457	457	457

Table 6.3: Importance distribution for commit events in PAID for Formula 4.3.

	$(i = 0.1, f = 0.9)$	$(i = 0.5, f = 0.5)$	$(i = 0.9, f = 0.1)$
0.10	6320	0	0
0.20	0	6340	0
0.30	0	0	6320
0.33	284	284	284

Table 6.4: Importance distribution for commit events in STARS for Formula 4.3.

Formula 4.3

When Formula 4.3

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ proximity(I_i, e) & : \quad e \in I_i \subseteq R_u \end{cases}$$

is used, the model weight factors are taken into account. However, this formula does not work very well in our environment, where most users have a number of overlapping interests. In that case, the application of this formula leads to an arbitrary choice of the *first* interest and thus results in an equally arbitrary importance value. The importance distributions for commit events are shown in Tables 6.3, 6.4 and 6.5.

	$(i = 0.1, f = 0.9)$	$(i = 0.5, f = 0.5)$	$(i = 0.9, f = 0.1)$
0.08	2887	2887	2887
0.10	822	0	0
0.20	0	822	0
0.30	0	0	822
0.33	11	11	11

Table 6.5: Importance distribution for commit events in TRAMP for Formula 4.3.

	$(i = 0.1, f = 0.9)$	$(i = 0.5, f = 0.5)$	$(i = 0.9, f = 0.1)$
0.10	6320	0	0
0.20	0	6340	0
0.30	0	0	6320
0.33	284	284	284

Table 6.6: Importance distribution for commit events in STARS for Formula 4.4

	$(i = 0.1, f = 0.9)$	$(i = 0.5, f = 0.5)$	$(i = 0.9, f = 0.1)$
0.08	2855	2855	2855
0.09	32	0	0
0.10	822	0	0
0.14	0	32	0
0.19	0	0	32
0.20	0	822	0
0.30	0	0	822
0.33	11	11	11

Table 6.7: Importance distribution for commit events in TRAMP for Formula 4.4.

Formula 4.4

When Formula 4.4

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ \frac{\sum_{i=1}^{m} proximity(I_i, e)}{m} & : \quad e \in I_i \subseteq R_u \end{cases}$$

is used, all (overlapping) interests that match an event are taken into account. While one could expect major differences in importance distribution when comparing this formula to the previous one, the differences are minor. In fact, the importance distributions for commit events in STARS are identical for Formula 4.3 (Table 6.4) and Formula 4.4 (Table 6.6). For TRAMP, the distribution is not identical, but very similar, as shown in Table 6.7. For PAID, all 3548 notifications are assigned the value 0.33.

Early user feedback indicated that users found this formula non-intuitive due to the ability of several low importance events to degrade the relevance of an single overlapping high importance event.

Formula 4.5

When Formula 4.5

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ \max(proximity(I_i, e)) & : \quad e \in I_i \subseteq R_u \end{cases}$$

is used, the maximum importance value is selected. While the choice of the maximum may appear to be more reasonable than the choice of a somewhat arbitrary value (Formula 4.3) or the average (Formula 4.4), the actual impact on the importance distribution minimal.

For all three projects (PAID, STARS and TRAMP), the distribution for Formula 4.5 is identical to the distribution for Fomula 4.3 (Table 6.3, Table 6.4 and Table 6.5).

Formula 4.6

When Formula 4.6

$$r : e \mapsto r(e) = \begin{cases} 0 & : \quad e \notin R_u \\ \min(\sum_{i=1}^{m} proximity(I_i, e), 1) & : \quad e \in I_i \subseteq R_u \end{cases}$$

is used, overlapping relevance values add up when interests overlap. Thus, the resulting importance distributions differ significantly. Figures 6.5, 6.6 and 6.7 display these distributions for PAID, STARS and TRAMP respectively.

We observe that the distributions become more continuous, compared to the distributions presented in the previous sections (which can be described and more discrete). Also, more notifications are rated with relevance values greater then 0.5, which combined with the more continuous distribution can be expected to make it easier for users to define individual thresholds.

The model weight factors influence the minimum importance values assigned to notifications. Also, the more relevance is assigned to the physical hierarchy metric (model weight factor component f), the more the distribution becomes shifted towards higher values.

For PAID and STARS, this results in more and more notifications that are rated with a relevance above 1.0. Thus for these projects, the obvious choice for model weight factors appears to be ($f = 0.1, i = 0.9$). For TRAMP, with most relevance values being quite small, this effect carries less improtance, because, even in worst case (model weight factors ($f = 0.9, i = 0.1$) only small number of notifications — 79 of a total of 3720 commit notifcations — are pushed beyond the 1.0 boundary.Still, for two out of three projects, the model weight factors apparently leads to more reasonable distributions.

Thus, we select the model weight factors ($f = 0.1, i = 0.9$) in combination with Formula 4.6 for our further discussion.

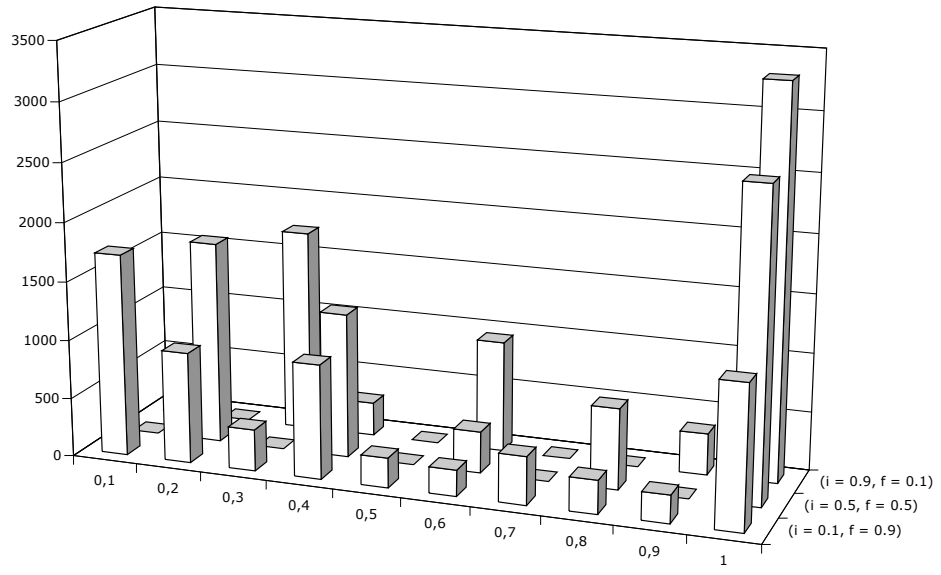

Figure 6.5: The distribution of importance values in notifications regarding CVS commit commands in PAID for Formula 4.6.

Project Comparison

In this section, we concentrate on Formula 4.6 and use it in combination with the model weight factors ($f = 0.1, i = 0.9$). We compare our three projects in terms of commit, post and read events.

First, we look at the notifications that are triggered by CVS "commit" commands. In Figure 6.8 we see the graphical representation of the importance distributions for the three projects PAID, STARS and TRAMP.

We observe that for all projects, most notifications have small importance values (less than 0.5) and the number of notifications becomes smaller, the greater the importance values become. In STARS und PAID, there is also a quite large number of notifications with importance values equal or greater than 1.0.

One possible explanation would be that the level of collaboration (multiple developers working collaboratively on a set of files) was higher in STARS and PAID. However, one big difference between STARS and PAID on the one hand and TRAMP on the other was the project duration. As the testbed does not expire interests, long running projects can be expected to produce

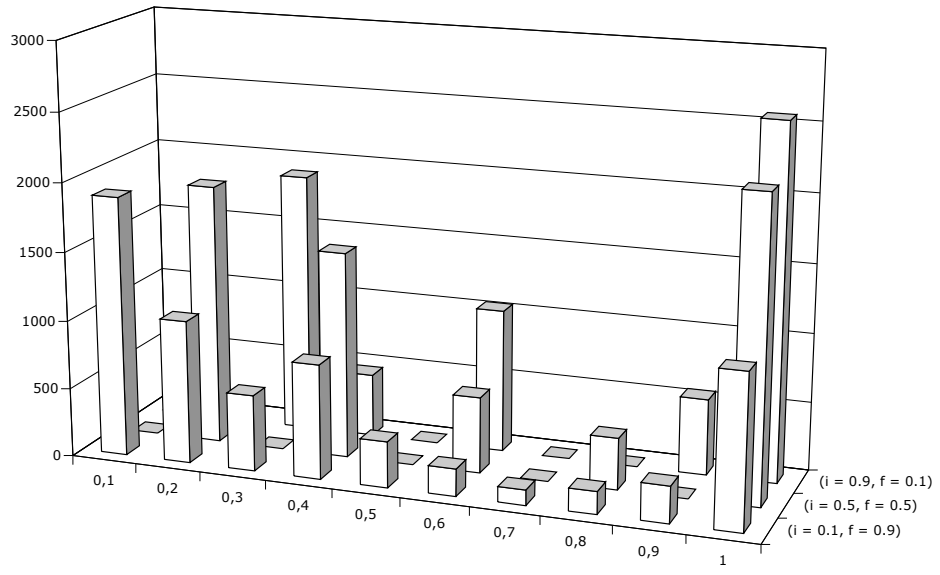

Figure 6.6: The distribution of importance values in notifications regarding CVS commit commands in STARS for Formula 4.6.

a lot of high-importance notifications "late" in the project when simulated. STARS and PAID lasted approximately three times as long as TRAMP.

The distribution of LOTUS NOTES post and read commands is different: due to the simpler design of the ABXNotesModel (employing a simple thread hierarchy based metric), the importance values for both types of notifications are multiples of 0.3 (Figures 6.9 and 6.10).

In Figure 6.9 we can observe that most notifications related to LOTUS NOTES post commands get the importance value of 0.3 — except in the PAID project, where a large number of notifications received the importance value of 0.6. The higher value results from the calculation $2 \cdot 0.3$ that happens when a message is posted in a thread where a user has posted two messages: in that case, two interests match — each resulting in a importance of 0.3 — and the resulting importance values are then combined using our cumulative formula.

When a user has posted more than two messages, e.g. n, then the relevance assessment results in the value $n \cdot 0.3$, which leads to the values 0.9 and above. However, the cumulative formula cuts high importance values off

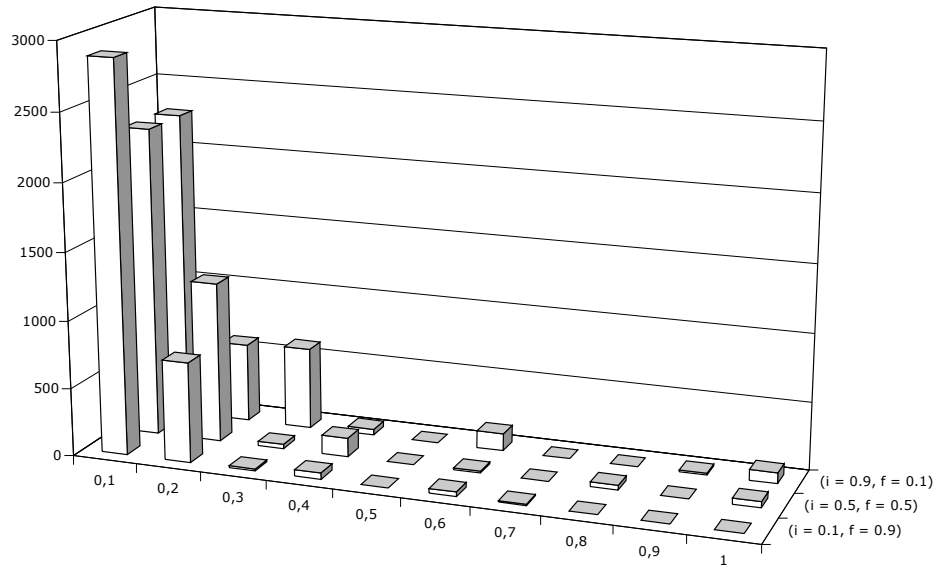

Figure 6.7: The distribution of importance values in notifications regarding CVS commit commands in TRAMP for Formula 4.6.

at 1.0, so in our distribution diagrams, we can at most observe 1.0.

The significant number of post notifications with importance 0.6 in PAID indicates a high number of discussion threads where a user has posted two messages. In STARS and TRAMP, most threads would contain at most one message from each user of the system. This means that in PAID postings spawned discussions that motivated the original poster to post additional messages.

A similar discrete distribution can be derived for the LOTUS NOTES read commands (Figure 6.10)[3].

6.2.7 Conclusion

The retrospective experiments have shown that any awareness system needs an Relevance Assessment service to manage the volume of events that occur

[3]While we can observe a large number of notifications with values other than 0.3 and 0.6 in the digram, namely 0.4 and 0.7, these values are actually the result of rounding issues in lower layers of our system and correspond to actual values 0.33 and 0.62 respectively

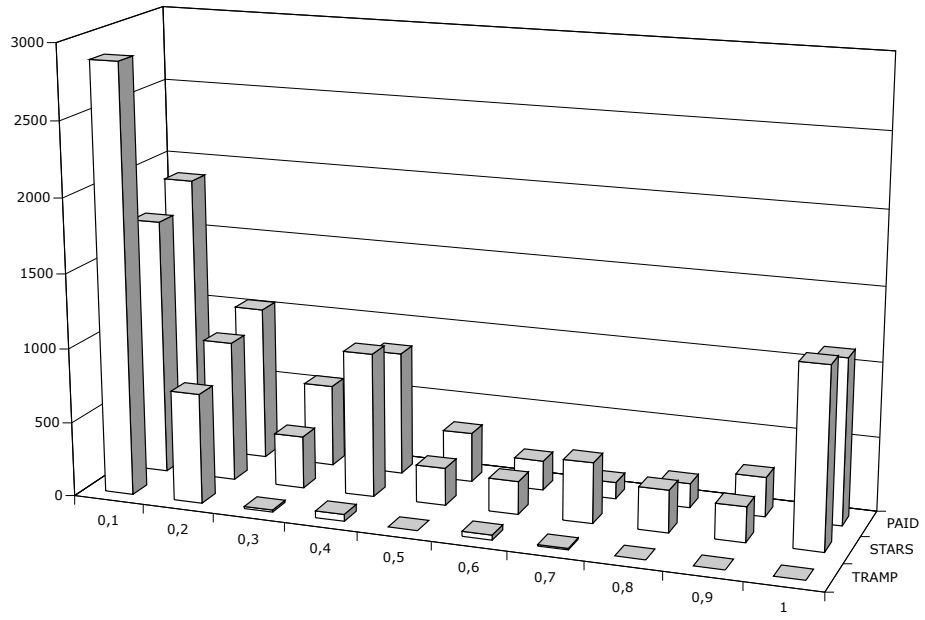

Figure 6.8: The distribution of importance values in notifications regarding CVS commit commands. While PAID and STARS took more than three semesters to complete, TRAMP lasted only one semester. This is one possible explanation for the large number of notifications with importance values above 0.9: as the testbed does not expire interests, long running projects can be expected to produce a lot of high-importance notifications "late" in the project when simulated.

on a typical project.

We also found that our initial setup of just two rules might not be sufficient. As most events in our projects are LOTUS NOTES read events, we should try to either lower the importance these events or even reduce their number, e.g. by limiting the lifetime of the respective interests. At the same time, the number of LOTUS NOTES post events could be higher, and we should therefore increase their number by setting individual interests for at least one's team message board.

We observed that these projects will profit the most from using our awareness system that have a "compatible" software configuration management policy, i.e. the developers check-in their source code early and often. Then, if there is overlap, the developers can reap the benefits of increased awareness

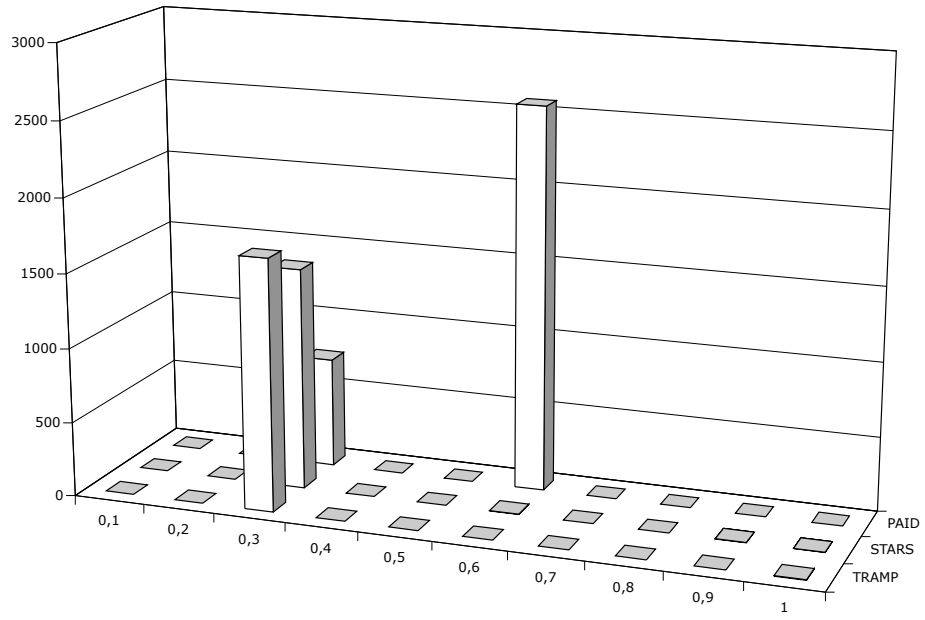

Figure 6.9: The distribution of importance values in notifications regarding LOTUS NOTES post commands. The significant number of post notifications with importance 0.6 in PAID indicates a high number of discussion threads where a user has posted two messages.

resulting from using AWARENESS BUILDER.

The distribution of importance values within the notifications of a project appears to be reasonable when using model weight factors ($f = 0.1, i = 0.9$) in combination with Formula 4.6. As most notifications have small importance values, the users can use the individual importance threshold feature of the AWARENESS BUILDER client to effectively distinguish between events with average importance and events with high importance (e.g. by setting the individual threshold to 0.5).

However, the retrospective experiments do not provide the answer to the question whether the importance weights assigned to notifications automatically by our system really reflect the importance as perceived by the users and whether the system is usable at all. Therefore, we now present the results of our prospective experiments with the ARENA project, where we tried to validate these questions as well as the general usability of our system.

simulation and data from the actual project.

Finally, in ARENA, the interests eventually expire, while in a simulated project they would not. We have simulated the projects in an accelerated fashion, but did not change the ABXD server to run the clock faster when simulating.

Early user feedback motivated us to use the cumulative formula to combine importance values from multiple interests into one and to set the global threshold to 0. It also led to putting a high weight on the inclusion hierarchy metric in the ABXCVSModel and on the thread hierarchy metric in the ABXNotesModel.

In the following section we compare the data obtained from the deployment of AWARENESS BUILDER in the actual ARENA project with the data from the simulations of PAID, STARS, TRAMP and (simulated) ARENA. Then, we will look at the results of a survey among ARENA participants that reveals how the users rated our system.

6.3.2 ARENA Project Log Analysis

The ARENA project was small in terms of participants when compared with the older projects PAID, STARS, and TRAMP (Table 6.8). During the course of the project, about 20 thousand events where generated that triggered about 40 thousand notifications. The number of events per participants was below average — however, the number of notifications per participants was significantly above average. This resulted in a comparatively low filtering ratio of 94,7% — and in as much as much as two notification triggered by each event on the average.

The high number of notifications was partly due to the individual interests that users would create manually during the project in addition to those generated automatically. This effect outweighs the fact that interests — both manually and automatically created ones — would typically expire after one week. Without these effects, the number of notifications would have been approximately 25% less, namely 29.759 (Table 6.9) and the notification-based metrics such as Notifications/Participants, Filtering Ratio and Notifications/Events much more inline with the metrics of the other simulated projects.

Still, the filtering ratio in ARENA was been quite low even without the influence of manually created interests. Table 6.9 shows that even without these additional interests, each event has triggered on the average 1,3 notifications, the highest value of the surveyed projects.

The individual interests have also led to a different distribution of notifications between the three event types supported by our system (CVS commit,

	PAID	STARS	TRAMP	ARENA
Logged Participants[4]	110	**121**	78	37
Final Lines of Code	**231.601**	58.435	25.071	16.546
Final Code Files	**1.673**	611	269	121
Events	**85.948**	60.136	58.223	20.712
Notifications	**92.919**	54.691	69.418	40.854
Events*Participants	**9.458.240**	7.276.456	4.541.394	766.344
Events/Participants	**782**	497	746	560
Notifications/Participants	845	451	890	**1.104**
Filtering Ratio $(1 - \frac{Notifications}{Events*Participants})$	**99,0%**	99,2%	98,5%	94,7%
Notifications/Events	1,1	0,9	1,2	**2.0**

Table 6.8: Input and Output Data Volume (bold denotes maximum values for each row).

	Simulated ARENA	Actual ARENA
Participants	37	37
Final Lines of Code	16.546	16.546
Final Code Files	121	121
Events	20.712	20.712
Notifications	26.730	40.854
Events*Participants	766.344	766.344
Events/Participants	560	560
Notifications/Participants	722	1.104
Filtering Ratio	96,5%	94,7%
Notifications/Event	1,3	2,0

Table 6.9: Manual interests created during the ARENA project increased the number of notifications, lowering the filtering ratio.

		CVS commit			LOTUS NOTES post		LOTUS NOTES read
		Frac.	No.	Frac.	No.	Frac.	No.
PAID	Events	8,3%	7.171	9,8%	8.409	81,9%	70.404
(Simulated)	Notifications	7,0%	6.505	3,6%	3.312	89,4%	83.102
STARS	Events	7,0%	4.181	8,4%	5.065	84,6%	50.890
(Simulated)	Notifications	12,1%	6.604	2,7%	1.493	85,2%	46.594
TRAMP	Events	1,0%	580	7,6%	4.413	91,4%	53.230
(Simulated)	Notifications	5,4%	3.720	2,4%	1.693	92,2%	64.005
ARENA	Events	10,0%	2.073	8,1%	1.682	81,9%	16.957
(Simulated)	Notifications	35,7%	9.549	3,0%	802	61,3%	16.379
ARENA	Events	10,0%	2.073	8,1%	1.682	81,9%	16.957
(Actual)	Notifications	26,0%	10.605	7,2%	2.937	66,9%	27.312

Table 6.10: LOTUS NOTES read events are the most frequent event type.

LOTUS NOTES post, and LOTUS NOTES read).

In Table 6.10 we see that CVS commit related notifications are over-represented when we compare ARENA to the other projects.

Despite having half the participants, ARENA witnessed almost four times more check-ins than TRAMP. This was probably due to ARENA having a software configuration management policy that encouraged frequent check-ins. This certainly helped ARENA to benefit from AWARENESS BUILDER more than TRAMP participants would had benefited. Unfortunately, we cannot tell whether this software configuration management policy was inherent to the ARENA project, or whether it was an effect of AWARENESS BUILDER usage.

The (absolute) number of commit notifications in the real project is only slightly higher than the number of these notifications in the simulated one. This indicates that users did not have to create additional interests and that the automatic interest generation worked quite well in this case. Informal user feedback received during the project supports this assumption.

On the other hand, the number of post notifications is much larger (almost four times as large) in the actual project than in the simulated one. Obviously, users created interests manually for post events. In Section 6.3.3 we will see evidence that users preferred to monitor the activities of their own team rather than the activities of the other teams. As the AWARENESS BUILDER client made it easy to set an interest for activities on a given message board, we assume that the large number of post notifications comes from manual interests set for post actions on the user's own team message

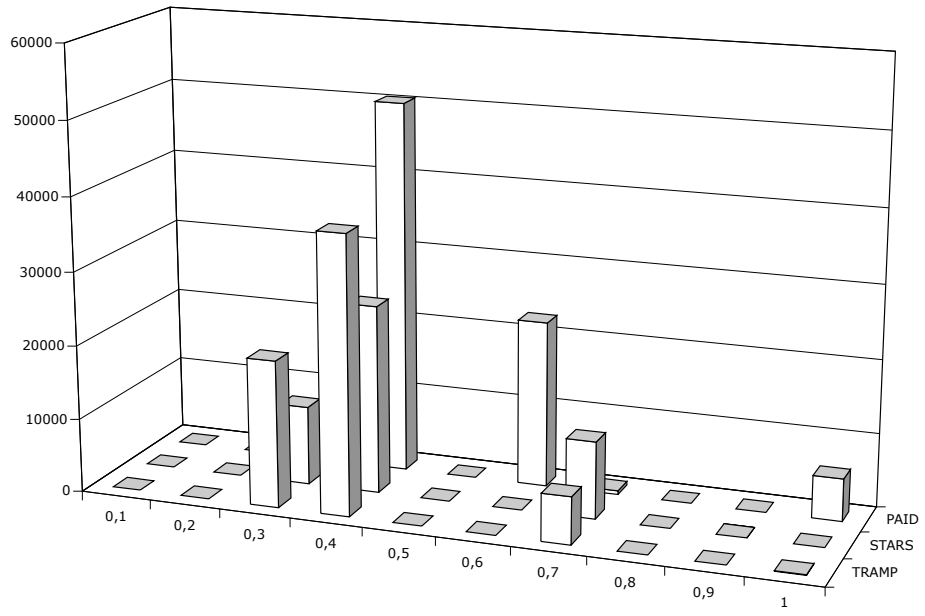

Figure 6.10: The distribution of importance values in notifications regarding LOTUS NOTES read commands. In the prospective experiment, we discovered that end users preferred lower importance ratings for most read notifications.

6.3 Prospective Experiment

6.3.1 Initial Setup

For the ARENA project, we deployed the AWARENESS BUILDER system in a similar way as in the retrospective experiments. However, the initial set of rules and interests did not remain static, but was allowed to change during the course of the project through user configuration. ARENA participants could add and modify their own interests, while we did not allow to add and modify the rules for the sake of making the learning curve for the tool shallower (we discovered that users not familiar with the system had a hard time when trying to understand the implication of rules).

Another difference between the simulated projects and the actual ARENA case study was the fact that in the retrospective experiments the CVS repository was in its final state when the simulation was performed. During the ARENA project, the repository was filling with source files and changing over time — this leads to apparent inconsistencies when comparing data from the

board.

The effect of the interest expiry feature can also be observed on the distribution of importance values in notifications that regard the CVS commit command (Table 6.11): overall, the number of notifications is less in almost all importance categories. One exception is the smallest importance value category (0,1), in which there are more notifications in the real project than in the simulated one. This has to result from additional individual interests created manually. Also, such manual interests could explain the peak of notifications at value 0,6.

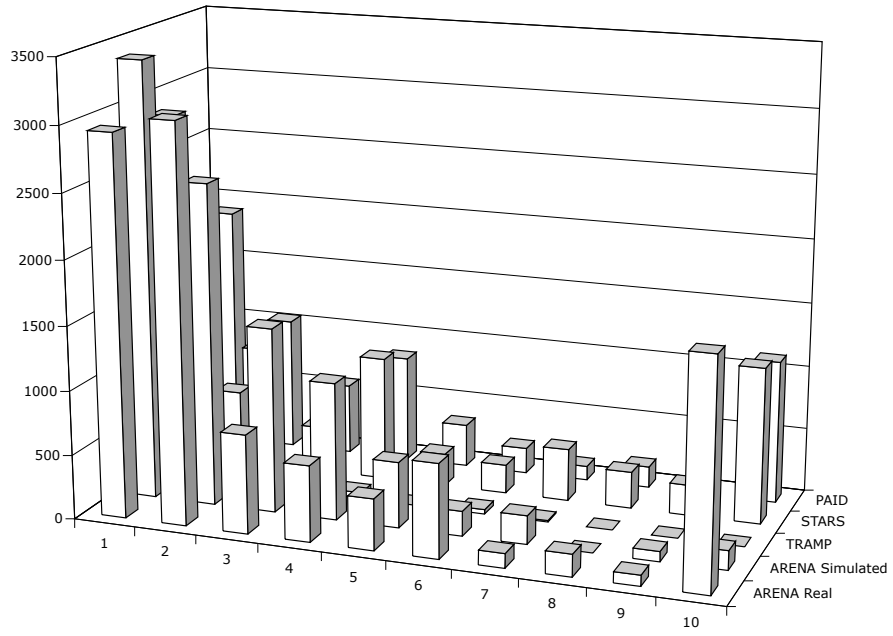

Figure 6.11: The distribution of importance values in notifications regarding CVS commit commands — including ARENA. In the actual ARENA project, end user's created additional interests, increasing the number of notifications with importance values above 0.9.

The effect of additional individual interests becomes more obvious, when we observe the distributions of notifications related to LOTUS NOTES post and read commands (Figures 6.12 and 6.13).

For post notifications the effect of the interest expiry feature cannot be observed. Instead we can see the effect of additional manual interests, which increased the overall number of post notifications significantly. Also, the

distribution of importance values has become more even when compared with the rather discrete distribution that resulted from the simulation (in the simulated projects, the values were limited to multiples of 0.3).

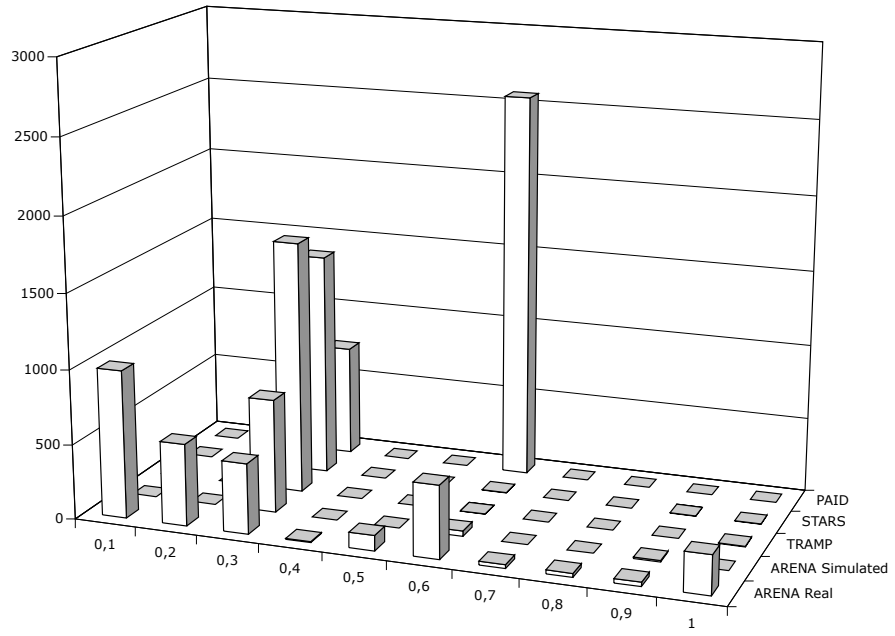

Figure 6.12: The distribution of importance values in notifications regarding LOTUS NOTES post commands. In the actual ARENA project, end users created manual interests for postings on their own teams's forum.

A similar effect can be observed when looking at the distribution of importance values that regard the read commands (Figure 6.13). Again, the additional manual interests have triggered more notifications and these notifications out-weigh any effect that expiring interests might have had. And also, the distribution of importance values has become quite even.

The decision to disallow the modification of rules prevented users from getting less LOTUS NOTES read related notifications. However, we observed that users would work around this limitation of our deployment setup by setting their interests with regard to their individual importance threshold.

With this section we conclude the quantitative analysis of the system. Next we discuss the qualitative results from asking the users about their opinion on our system.

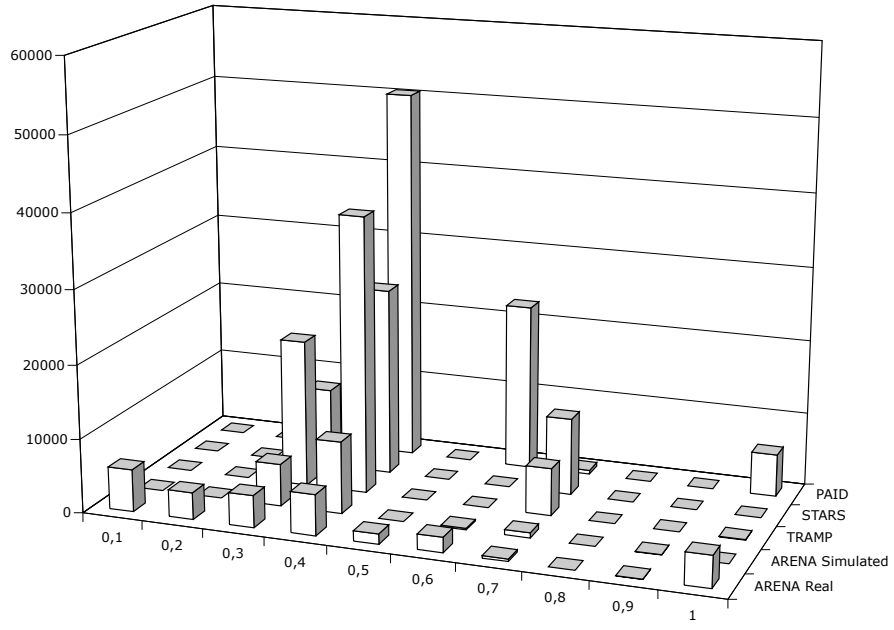

Figure 6.13: The distribution of importance values in notifications regarding LOTUS NOTES read commands. The decision to disallow the modification of rules prevented users from getting less LOTUS NOTES read related notifications. However, we observed that users would work around this limitation of our deployment setup by setting their interests with regard to their individual importance threshold. Again, end user's created manual interests for certain discussion threads.

6.3.3 ARENA Course Evaluation Questionaire

Each time a project course ends, the students are given the opportunity to provide feedback. This is usually done by means of an online questionnaire. The ARENA questionnaire contained 90 questions covering a wide range of topics that covered the course material, the presentation of material, the course environment, the structure of the course, the communication media, the technology, the time spent in the course and others. Out of these 90 questions, 12 questions were specific to the AWARENESS BUILDER system (Table 6.11).

The students were given one week to answer the questions. As filling out the questionnaire was not mandatory for getting the credits for the course, we received 11 out of 18 questionnaires. This number corresponds to about

How often did you use ABX?	always	0	1	2	3	5	never	
How did you like the user interface?	extremely	1	0	7	2	1	not at all	
How did you like the interests/rules system?	extremely	0	2	6	2	1	not at all	
How useful did you find ABX for your work in ARENA?	useful	1	2	4	0	4	useless	
How often did an ABX notification trigger a communication between you and a fellow student?	very often	0	3	3	0	5	never	
You used abx to monitor the activities of	your team:	9	other teams:		0	both:	2	
You preferred to monitor	NOTES:	5	CVS:		3	both:	3	
Did ABX prevent a CVS conflict from occuring?	yes:	2	no:		5	never had a conflict:	4	
List three things you liked about ABX								
List three things you hated about ABX								

Table 6.11: AWARENESS BUILDER specific questions from the ARENA course evaluation questionnaire. Numbers indicate respondant answers: e.g. one respondant liked the user interface extremely well, 7 found it average, two found it poor, and one disliked it.

60% of the students who took this course (previously, we listed 37 participants for ARENA — that number included also the instructors, coaches and administrators of the course, who did not participate in the evaluation questionnaire).

The ABX specific part of the questionnaire consisted of ten multiple-choice and two open questions. The first five multiple-choice questions allowed a spectrum of answers, while three of them asked for a selection between three options. In Table 6.11 we have printed the frequency of occurrence for each possible choice. Next, we want to discuss the multiple-choice questions, and then move to some example answers we received to the open questions.

First, we notice that 5 users did not use the tool. This can an indication for usability issues, but can also be a result of reliability problems we had with ABX during early deployment. Also, users gave medium scores for the user interface and rule and interest system that is the visible part of the

Relevance Assessment service. However, the majority of the students did find the system useful for their work in ARENA (although there are four students who did not find it useful at all).

An interesting result is that the majority of students experienced a situation where our system triggered a communication — this provides evidence to the common claim that awareness systems are an important communication facilitator in distributed environments.

Most users used our system to monitor the activities of their own team rather then the activities of other teams. This is an indication that the organizational partitioning in ARENA worked and that team members focused on their own teams work. Also, this also indicates what kind of individual interests the users created, namely interest that would allow them to monitor their own team's message board, their own team's part of the CVS repository, and the actions of the members of their team.

Somewhat surprisingly, most users preferred to monitor the communication in LOTUS NOTES rather than just the activities in the CVS repository. However, in at least two cases, our system prevented a CVS conflict from occurring.

We received six answers to our ABX related open questions, although two students just stated that they have not used our system at all.

One student has expressed his wish to have the presentation of awareness information integrated in the IDE rather than in a separate client. Also, the same student has suggested to use our system right from the beginning (we introduced the tool a few weeks after the course started).

Two students mentioned the stability problems we had with the system. Also, one student found that despite the work done by the Relevance Assessment service, he would still get too many notifications that are not interesting for him. One reason for this — we assume — could be our decision to not allow modifications to the initial two rules.

6.4 Conclusion

The retrospective studies proved that real projects generate a number of events that cannot be forwarded to the participants without exposing them to a massive information overload. Awareness systems must include a Relevance Assessment service that reduces the number of notifications a user receives by ranking the events according to the users interest profile.

The prospective study provided evidence our awareness system can be a welcome addition to the toolset of a developer working in a distributed environment. Our prototype implementation, AWARENESS BUILDER, indicates

that it is useful and can be deployed in controlled environments in spite of its experimental nature.

Chapter 7

Future Directions

The studies described in the Chapter 6 provide evidence that the prototype implementation, AWARENESS BUILDER is useful for developers working in a distributed environment.

It has still a number of shortcomings which fall into one of two categories: user interface and relevance assessment. In the area of user interface there there are several issues that need to be addressed in the future.

First, there are still improvements to the graphical user interface (GUI) that can be made to the existing AWARENESS BUILDER client, as the user interface has been rated mostly average.

Usability engineering techniques can be used to improve the user experience with the current client application. Nielsen [53] suggests that such improvements can be done with little effort with a selection of usability engineering methods that he calls *discount usability engineering*. Thus, this methodology might be well-suited for for projects with limited human-computer interaction (HCI) budgets.

Second, the implementation should be changed to allow easy integration of external presentation clients, such as ambient displays, tools (e.g. IDEs as one of the ARENA students suggested), web applications or chat systems.

This can be achieved by abandoning the current protocol used for communication between the AWARENESS BUILDER client and the ABXD and moving to standard protocols. Protocols already used in the Collection service, such as XML-RPC and SOAP, could be evaluated for their suitability for the Dissemination service, as well as special purpose information dissemination protocols such as RSS[1][35].

Third, the user interface could be radically redesigned using advanced visualization techniques, such as those applied in the TOWER project [66].

[1]RSS is an ambiguous acronym that stands for either RDF Site Summary, Rich Site Summary, or Really Simple Syndication

However, careful evaluation is needed to provide evidence that advanced visualization leeds to improvements in usability.

In the area of relevance assessment there is much work to be done. First, while the current implementation works, user feedback suggests that the quality of relevance assessment leaves much room for improvement.

The Relevance Assessment could also be redesigned to allow even easier changes to the algorithms used, e.g. at runtime. This would allow easy experimentation with different rules, interests and algorithms for combining interests. The existing implementation already provides an interface for distance models that supports this requirement. Similar interfaces could be designed for other parts of the Relevance Assessment service as well[2].

Also, relevance assessment is an activity that consumes a lot of processing time. Currently, not much attention was paid to the performance of the system (as this was not a primary design goal for the system). However, our experiments showed that the longer the system runs, the more interests exist and need to be matched against each incoming event. Long running projects with a large number of users can cause a significant load for the awareness server (also see Section 6.2.3). While we assume that this is not much of an issue when deploying the system in an actual project, faster processing would enable faster execution of experimental cycles for retrospective experiments.

We suggest that after the redesign of the Relevance Assessment service special attention is applied to these performance issues. Our quantitative evaluation method can be used here to establish a repeatable scenario that can be used for performance measurements.

Then, a new series of quantitative experiments should be conducted. Using our evaluation method and the testbed, the individual experiments can be easily repeated with the same set of input data from one particular project and the results of each experimental iteration can be used to compare the algorithms used. For this purpose, this dissertation is accompanied by a CD which contains the full source code for our system, as well as the complete set of data files. This material is available on request from the author[3].

Besides evolutionary steps to improve ABX, we envision that future work should also try radically different approaches to the design of awareness systems, as well. First, an awareness system similar to ABX can built today using a framework for routing XML messages, such as Jabber [42]. Second, we believe that there are similarities between awareness systems and sophisticated debugging system, which we mentioned briefly in Section 4.4.5. Both

[2]e.g. in the existing implementation, the relevance functions can be selected during startup of the system via a preferences file, but are otherwise hard-coded into the system.

[3]As no effort has been made so far to anonymize the project data, the material cannot be made public on the Internet, yet.

system families deal with dynamic systems (for our discussion of similarities of organizations and software systems see Section 3.4) and can be therefore expected to share description languages for their data (i.e. events and notifications), as well as structure and behaviour.

Also, we believe that awareness systems have a lot of characteristic that make them suitable for certain project management tasks. Thus, future work should research usage scenarios in project management, where awareness systems could substitute tedious progress reporting procedures and help controlling distributed software development projects. However, the benefits of such usage scenarios have to be carefully weighted against their possible negative impact on the end user acceptance.

All designers of awareness systems must not forget about privacy concerns that such systems raise. In this work, we have only scratched the surface of the privacy issues involved and have suggested solutions that are workable for research systems. We believe that careful experimentation is needed to find out how much privacy people will voluntarily give up in order to reap the benefits of increased awareness. Designers of awareness system should try to approach the border of acceptable publicity but must be careful not to cross it.

Bibliography

[1] July 2002. http://www.xml-rpc.com.

[2] July 2003. http://wwwbruegge.in.tum.de/projects/arena/.

[3] Jan. 2003. http://www.google.com.

[4] Jan. 2003. http://www.cs.fsu.edu/~engelen/soap.html.

[5] Nov. 2004. http://www.m-w.com.

[6] America Online, Inc, July 2002. http://aim.aol.com.

[7] Apple Computer, Inc., July 2002. http://www.apple.com/ichat.

[8] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of Principles of Database Systems*, 2002.

[9] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[10] K. Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77, 1999.

[11] S. A. Bly, S. R. Harrison, and S. Irwin. Media spaces: bringing people together in a video, audio, and computing environment. *Communications of the ACM*, 36(1):28–46, 1993.

[12] B. W. Boehm. Improving software productivity. *Computer*, 20(9):43–57, Sept. 1987.

[13] M. Boyle, C. Edwards, and S. Greenberg. The effects of filtered video on awareness and privacy. In *Proceeding of the ACM 2000 Conference on Computer supported cooperative work*, pages 1–10. ACM Press, 2000.

[14] F. P. Brooks, Jr. *The Mythical Man Month.* Addison–Wesley, anniversary edition, 1995.

[15] B. Bruegge. A portable platform for distributed event environments. In *Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging*, 1991.

[16] B. Bruegge and A. H. Dutoit. *Object-Oriented Software Engineering: Conquering Complex and Changing Systems.* Prentice Hall, 2000.

[17] B. Bruegge, A. H. Dutoit, R. Kobylinski, and G. Teubner. Transatlantic project courses in an university environment. In *Seventh Asian Pacific Software Engineering Conference (APSEC 2000)*, 2000.

[18] M. Bürger. *Unterstützung von Awareness bei der Gruppenarbeit mit gemeinsamen Arbeitsbereichen.* PhD thesis, Institut für Informatik, TU München, 1998.

[19] D. N. Card, V. E. Church, and W. W. Agresti. An empirical study of software design practices. *IEEE Transactions on Software Engineering*, 12(2):264–271, Feb. 1986.

[20] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11), Nov. 1988.

[21] R. L. Daft and R. H. Lengel. Information richness: A new approach to managerial behaviour and organization design. In L. Cummings and B. Staw, editors, *Research in Organizational Behaviour.* JAI Press, 1984.

[22] R. L. Daft, R. H. Lengel, and L. K. Trevino. Message equivocality, media selection and manager performance: implications for information systems. *MIS Quarterly*, 11(3):355–366, 1987.

[23] H. Dietl. *Institutionen und Zeit*, volume 79 of *Die Einheit der Gesellschaftswissenschaften.* J. C. B. Mohr, 1993.

[24] P. Dourish. private communication, Feb. 2003.

[25] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the conference on Computer-supported cooperative work*, pages 107–114. ACM Press, 1992.

[26] P. Dourish and S. Bly. Portholes: supporting awareness in a distributed work group. In *Conference proceedings on Human factors in computing systems*, pages 541–547. ACM Press, 1992.

[27] A. H. Dutoit, O. Creighton, G. Klinker, R. Kobylinski, C. Vilsmeier, and B. Bruegge. Architectural issues in mobile augmented reality systems: A prototyping case study. In *8th Asia-Pacific Software Engineering Conference (APSEC 2001)*, Dec. 2001.

[28] A. H. Dutoit, J. Johnstone, and B. Bruegge. Knowledge scouts: Reducing communication barriers in a distributed software development project. In *Asian Pacific Software Engineering Conference*, Dec. 2001.

[29] T. Erickson, D. N. Smith, W. A. Kellogg, M. Laff, J. T. Richards, and E. Bradner. Socially translucent systems: social proxies, persistent conversation, and the design of babble. In *Proceeding of the CHI 99 conference on Human factors in computing systems : the CHI is the limit*, pages 72–79. ACM Press, 1999.

[30] W. Gaus. *Dokumentations- und Ordnungslehre. Theorie und Praxis des Information Retrieval*. Springer, Berlin et al., 1995.

[31] S. Greenberg. Peepholes: low cost awareness of one's community. In *Proceedings of the CHI '96 conference companion on Human factors in computing systems : common ground*, pages 206–207. ACM Press, 1996.

[32] R. E. Grinter, J. D. Herbsleb, and D. E. Perry. The geography of coordination: Dealing with distance in R&D work. *ACM*, 1999.

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

[34] C. Gutwin and S. Greenberg. The effects of workspace awareness support on the usability of real-time distributed groupware. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 6(3):243–281, 1999.

[35] B. Hammersley. *Content Syndication with RSS*. O'Reilly, 2003.

[36] R. H. R. Harper. Why do people wear active badges? Technical report epc-1993-120, Rank Xerox Research Centre Cambridge Laboratory, 1993.

[37] L. E. Holmquist. private communication, Feb. 2003.

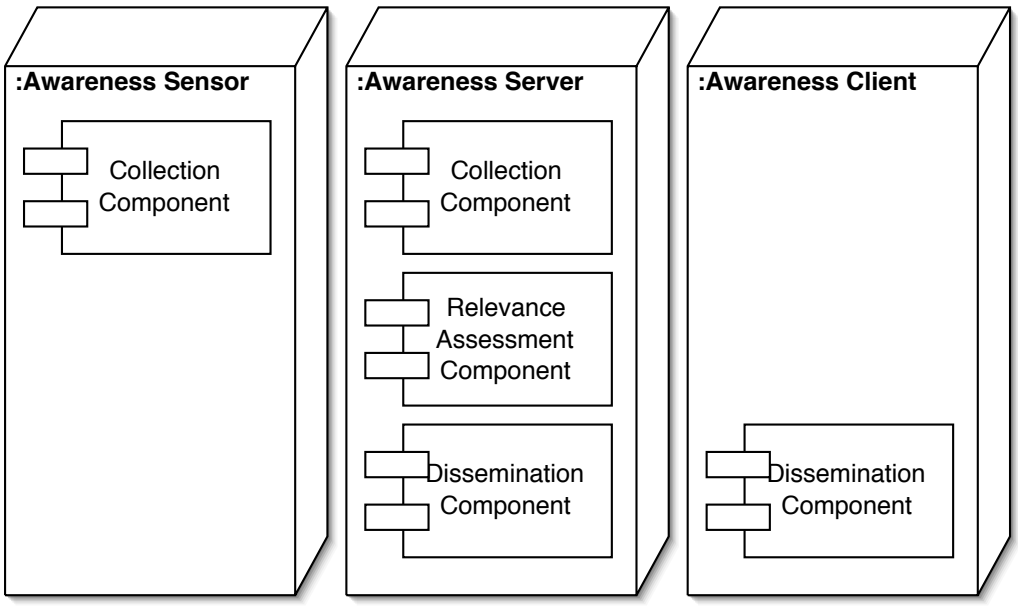

Figure 5.2: The collection component and the dissemination component are both distributed.

awareness system, but we have implemented it nonetheless to support the evaluation of the system.

Within the Awareness Server, the collection service and the relevance assessment service should be implemented in separate execution entities (tasks, threads, or processes), to allow for parallel processing. Otherwise, the collection service would have to wait for the relevance assessment component to finish its (possibly time consuming) computations before it could react to incoming events.

Thus, the awareness events pipe between the Collection Service and the relevance assessment service needs to behave like a buffer for which the reads and writes are synchronized — the collection service behaving like a Producer and the relevance assessment service behaving like a Consumer in terms of the standard Producer/Consumer paradigm.

This is also the case for the second pipe in the system, namely the awareness notifications pipe between the relevance assessment service and the dissemination service. Obviously, it is not desirable for the relevance assessment component to block until the users have been notified.

The ABX system consists of six components: two application sensors, two proxy sensors, a server component and a client application (Figure 5.4). The CVS and the LOTUS NOTES application sensors are incorporated into the

[38] E. M. Huang, J. Tullio, T. J. Costa, and J. F. McCarthy. Promoting awareness of work activities through peripheral displays. In *Conference Extended Abstracts on Human Factors in Computer Systems*, pages 648–649. ACM Press, 2002.

[39] ICQ, Inc, July 2002. `http://web.icq.com`.

[40] E. A. Isaacs, J. C. Tang, and T. Morris. Piazza: a desktop environment supporting impromptu and planned interactions. In *Proceedings of the ACM 1996 conference on Computer supported cooperative work*, pages 315–324. ACM Press, 1996.

[41] H. Ishii. private communication, Feb. 2003.

[42] Jabber.com, Inc, July 2002. `http://www.jabber.com`.

[43] M. Jarke and W. Prinz. Tower. Projektblatt.

[44] P. Jeffrey and A. McGrath. Sharing serendipity in the workplace. In *Proceedings of the third international conference on Collaborative virtual environments*, pages 173–179. ACM Press, 2000.

[45] R. Kraut, C. Egido, and J. Galegher. Patterns of contact and communication in scientific collaboration. In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '88)*, pages 1–12, Portland, Oregon, 1988. ACM Press.

[46] R. E. Kraut and L. A. Streeter. Coordination in software development. *Communications of the ACM*, 38(3):69–81, Mar. 1995.

[47] M. D. Langone. Large group awareness trainings. *Cult Observer*, 15(1), 1998.

[48] A. Lee, K. Schlueter, and A. Girgensohn. Sensing activity in video images. In *CHI'97 Extended Abstracts*, pages 319–320. ACM Press, 1997.

[49] M. Leliveld. Xwho, ein system zur visualisierung der benutzer in rechnernetzen. Fortgeschrittenenpraktikum, Insitut für Informatik, TU München, 1994.

[50] S. Maguire. *Debugging the development process.* Microsoft Press, 1994.

[51] Microsoft Corporation, July 2002. `http://messenger.msn.com`.

[52] J. Nielsen. *Usability Engineering.* AP Professional, 1993.

[53] J. Nielsen. Guerrilla hci: Using discount usability engineering to penetrate the intimidation barrier. In *Cost-Justifying Usability*, pages 245–272. Morgan Kaufmann, 1994.

[54] M. O'Hara-Devereaux and R. Johansen. *Globalwork: Bridging Distance, Culture and Time*. Jossey-Bass, 1994.

[55] J. Oikarinen and D. Reed. RFC 1459: Internet Relay Chat Protocol, May 1993. Status: EXPERIMENTAL.

[56] U. Pankoke-Babatz. "awareness: Spannungsfeld zwischen beobachter und beobachtetem". In *Workshop Von Groupware zu GroupAware der D-CSCW 1998*, pages 5–11. ACM Press, 1998.

[57] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, Dec. 1972.

[58] E. R. Pedersen and T. Sokoler. Aroma: abstract representation of presence supporting mutual awareness. In *conference proceedings on Human factors in computing systems*, pages 51–58. ACM Press, 1997.

[59] D. E. Perry, A. A. Porter, and L. G. Votta. Empirical studies of software engineering: a roadmap. In *Proceedings of the conference on The future of Software engineering*, pages 345–355. ACM Press, 2000.

[60] T. J. Peters and R. H. Waterman. *In search of excellence: Lessons from America's best-run companies*. Harper and Row, 1982.

[61] A. Picot, H. Dietl, and E. Franck. *Organisation: Eine ökonomische Perspektive*. Schäfer-Poeschel, 1997.

[62] A. Picot, R. Reichwald, and R. T. Wigand. *Die grenzenlose Unternehmung*. Gabler, 1996.

[63] B. Pinkerton. *WebCrawler: Finding what people want*. Phd thesis, University of Washington, Nov. 2000.

[64] W. Prinz. A day in life of a tower world user.

[65] W. Prinz. private communication, Apr. 2004.

[66] W. Prinz and U. Pankoke-Babatz. Tower — theatre of work enabling relationships. *ECRIM News*, (42):46–47, 2000.

[67] W. Prinz, U. Pankoke-Babatz, W. Gräther, T. Gross, S. Kolvenback, and L. Schäfer. Presenting activity information in an inhabited information space. In *Inhabited Information Spaces*, pages 181–208. Springer, 2004.

[68] G. Probst, S. Raub, and K. Romhardt. *Wissen Managen*. Gabler Verlag, Wiesbaden, 1997.

[69] T. Rodden. Populating the application: a model of awareness for cooperative applications. In *Proceedings of the ACM 1996 conference on Computer supported cooperative work*, pages 87–96. ACM Press, 1996.

[70] W. Scholl. Informationspathologien. In E. Frese, editor, *Handwörterbuch der Organisation*, pages 900–912. Schäffer-Poeschel, 3rd edition, 1992.

[71] L. Schäfer and S. Küppers. Camera agents in a theatre of work. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 218–219. ACM Press, 2002.

[72] T. Schümmer. *TUKAN - Entwicklung einer kooperativen Software-Konfigurations-Umgebung*. PhD thesis, TU Darmstadt, 1999.

[73] T. Skog, L.-E. Holmquist, and S. Ljungblad. Bringing computer graphics to everyday environments with informative art. In *Siggraph Conference 2002, Sketches and Applications*, 2002.

[74] I. Smith and S. E. Hudson. Low disturbance audio for awareness and privacy in media space applications. In *Proceedings of the third ACM international conference on Multimedia*, pages 91–97. ACM Press, 1995.

[75] G. Snelting. Paul feyerabend und die softwaretechnologie. *Software Tools for Technology Transfer*, pages 1–5, november 1998.

[76] M. Solomon. Spam wars: Spammer says it is almost impossible to stop, 2002. `http://maccentral.macworld.com/news/0211/14.spam.php`.

[77] S. St. Laurent, J. Johnston, and E. Dumbill. *Programming Web Services with XML-RPC*. O'Reilly, 2001.

[78] W. F. Tichy. Should computer scientists experiment more? 16 reasons to avoid experimentation. *IEEE Computer*, 31(5):32–40, 1998.

[79] W. F. Tichy, P. Lukowicz, L. Prechelt, and E. A. Heinz. Experimental evaluation in computer science: A quantitative study. *Journal of Systems and Software*, 28(1):9–18, 1995.

[80] D. Tidwell, J. Snell, and P. Kulchenko. *Programming Web Services with SOAP*. O'Reilly, 2001.

[81] M. Weiser and J. S. Brown. Designing calm technology. *PowerGrid Journal*, 1(1), 1996.

[82] E. Wischnewski. *Modernes Projektmanagement: eine Anleitung zur efktiven Unterstützung der Planung, Durchführung und Steuerung von Projekten*. vieweg, 4th edition, 1993.

[83] C. Wisneski, H. Ishii, A. Bahley, M. Gorbet, S. Braver, B. Ullmer, and P. Yarin. Ambient displays: Turning architectural space into an interface between people and digital information. In *Proceedings of the First International Workshop on Cooperative Buildings (CoBuild '98)*, February 1998.

[84] Yahoo! Inc., July 2002. `http://pager.yahoo.com`.

[85] Q. A. Zhao and J. T. Stasko. Evaluating image filtering based techniques in media space applications. In *Proceedings of the ACM 1998 conference on Computer supported cooperative work*, pages 11–18. ACM Press, 1998.