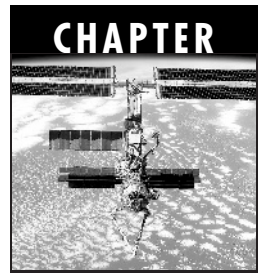

1 Introduction

New technologies have changed the ways in which people interact and collaborate over a distance. Users can stay connected over a network and practice new ways of collaborative working. Instead of working face-to-face most of the time, today many people collaborate with remote peers via the Internet. In professional work life, employees in distributed companies collaborate via distributed work groups, workers in distant parts of a virtual organization can form dynamic ad-hoc teams for a step in a production process, and people participate in virtual communities to increase their professional capabilities. This process is also visible in private life, where computer users increasingly participate in communities to make their lives easier or more interesting.

As a result, more and more applications are designed for use by more than one user. Domains in which this has become obvious are multi-player games, websites that foster interaction among visitors, applications for interaction between mobile users, systems that foster collaborative learning, interactive workspaces and smart environments, or peer-to-peer applications, to name only a few. In such areas we can see a shift in interest from human computer interaction to computer-mediated human interaction.

This book discusses how applications for supporting computer-mediated interaction or *groupware* applications can be built. A groupware application is a combination of software, hardware and social processes that supports groups in their interaction. The groupware thus is what mediates interaction in computer-mediated interaction.

We will use a *pattern language* to build such groupware systems. Pattern languages combine *patterns* from a specific application domain, and consist of patterns and the relations between these patterns. Patterns capture design knowledge and rationale so that a novice can make design decisions in the same way that experts would. By means of design patterns one can describe expert knowledge in the form of rules of thumb that helps to solve a problem in the application domain of the pattern language.

Depending on your role in the development process of a groupware system, you would make use of a pattern language in different ways:

— As a *software developer*, you would find guidance on how to implement groupware applications. The patterns would outline different functional components that need to be developed when approaching groupware-typical problems.

— As a *user*, the patterns would provide you with an idea on how the groupware applications might look and how social processes might change when groupware comes into play. Your role would probably require you to identify functional aspects of your envisioned groupware application and communicate them to the software developers. This can raise your level of participation and help to ensure that the solution fits into your context. In the case of high-level patterns, you might also be able to implement the pattern on your own by tailoring an existing groupware environment.

— Finally, as a *student* or *researcher* in the field of computer-supported collaborative work, you might use patterns to frame your research: a pattern language documents best practices that have evolved during recent years, and in the case of most patterns, provides you with links to research literature in which such practices have been discussed.

To complement the patterns, we will provide a common scenario that serves as an example of how to apply the pattern approach in a concrete case of groupware development. We have chosen the case of a distributed software development team that wants to make use of groupware technology to support the distributed development of a collaborative game engine better. We will tell the story of Paul Smith, who is the project leader of the collaborative game engine project. The story will tell you about how social interaction is intertwined with professional activities, and how Paul interacts with a global network of people to achieve his project's goals.

The fact that we have selected collaborative software development does not, however, mean that the application of this book is restricted to that domain. As you

will see in the scenarios, many of the issues that arise in the collaboration of distributed software developers are to the same degree relevant in other application domains such as distributed management, distributed product development, or distributed learning. You can consider this in the same way as an architect would understand the creation of houses: in our example we will build a metaphorical *office building* for software developers. But the skills required to create such a building can also be used to build a school or a building for car design engineers. It is you as end user who will introduce the domain-specific context of your group, while this book concentrates on groupware infrastructure.

1.1 Groupware: Systems that Support Computer-Mediated Interaction

A famous definition of the term groupware defines groupware systems as "intentional group processes plus software to support them" (Johnson-Lenz and Johnson-Lenz, 1981). This definition includes various aspects that we have to consider when designing groupware solutions:

- The core of the definition is the *group*. A group of users wants to interact using groupware. Naturally, the members of the group should play an important role in the design of groupware. The groupware design has the purpose of creating a solution that satisfies the user's needs. *End-user requirements* therefore have to be the central issue during groupware design.
- The group interacts in an *intentional group process*. The *interaction between people* thus needs to play an important role in the design of any groupware solution. It has to become clear who interacts with whom. How strict the intentional group process is must be considered, ranging from unplanned interaction in virtual communities up to formally structured workflows in a distributed workgroup.
- The process is supported by *software*. The fact that software is mentioned third here emphasizes that the software itself should be a supportive facility to ease the interaction between people. The software should be adapted to the users' needs to best fulfill its supportive role. At this point the *software developer* comes into play. As software supports the group process, the software developer should support the users in adapting or creating software that fits the process.

Compared to a focus on design, which has the goal of supporting the group in the manipulation of content, support for social group interaction needs a broader focus, that of the relationships between users. Tools for manipulation are in most cases used by one user (even in collaborative systems), so they affect the relationship between the user and shared artifacts. Social interaction, on the other hand, affects the relationships between users and needs to address issues like trust and privacy. In contrast to the design of tools for the manipulation of artifacts, which mainly affects human-computer interaction, the focus should thus be on human-computer-human interaction (HCHI). The design of tools therefore focuses on the interaction of the user with the artifact, and considers the human-human interaction as a marginal aspect.

To provide customized designs of groupware mechanisms, we have to make use of a design process that is flexible enough to adapt to the group's needs. Experiences with the design of single user applications have already shown that many software development projects fail because of requirements inadequacies (Dorfman, 1997).

In such cases, the customer is typically involved in the early stages of the project as a source of design requirements. This set of requirements is then implemented by the software developers and subsequently the customer assesses the result. However, if the requirements were not specified correctly, customers receive a product that does not match their needs. This means that requirements in the context of computer-mediated interaction must always address social aspects as well as technical aspects, which is why they are called *socio-technical requirements*.

Unfortunately, these socio-technical requirements are often less clear to the stakeholders involved in the development of groupware applications. Two factors make this part of groupware development difficult:

— While in single user tasks, such as word processing or image editing, only one actor interacts with an artifact, groupware needs to support the interaction of many users with each other. An interaction partner is thus not a technical, deterministic, artifact, but a non-deterministic human.

— Users are not as familiar with using these new opportunities for interaction compared with single-user applications.

The theory of socio-technical design views a community from two perspectives: the social system, including group processes, roles, and flow of information, and the technical system, which includes tools used within the community, such as IT infrastructure or buildings. From a socio-technical perspective these two systems are highly interrelated. A socio-technical perspective on groupware design has to be aware of three key aspects (Bikson and Eveland, 1996):

— It is difficult to predict the reciprocal effect of changes to either the social or the technical system.

— The process used to create the socio-technical system will affect the acceptance of the system.

— Both social and technical systems change over time.

The tools in the technical system, i.e. the software that supports intentional group processes (Johnson-Lenz and Johnson-Lenz, 1981), can be classified in many different ways. One popular way to classify groupware is to distinguish how it support groups. Teufel et al. (1995) introduced such a model and distinguish between three different main support functionalities:

1. *Communication* focuses on the information exchange between cooperating group members.
2. *Coordination* concentrates on coordinating group tasks.
3. *Cooperation* adds the ability to accomplish group goals to the above support functionalities.

As all main support functionalities start with the letter *C*, Borghoff and Schlichter (2000) later on called this approach *3C-classification*. In their initial proposal

Teufel et al. (1995) positioned the three main functionalities in a triangle to cluster groupware applications in *system classes* of common functionality, i.e. *communication systems*, *workflow management systems*, *shared information spaces*, and *workgroup computing systems*.

In contrast to the initial approach of Teufel et al. (1995), we propose a different approach. Figure 1.1 places well-known groupware applications in two-dimensional

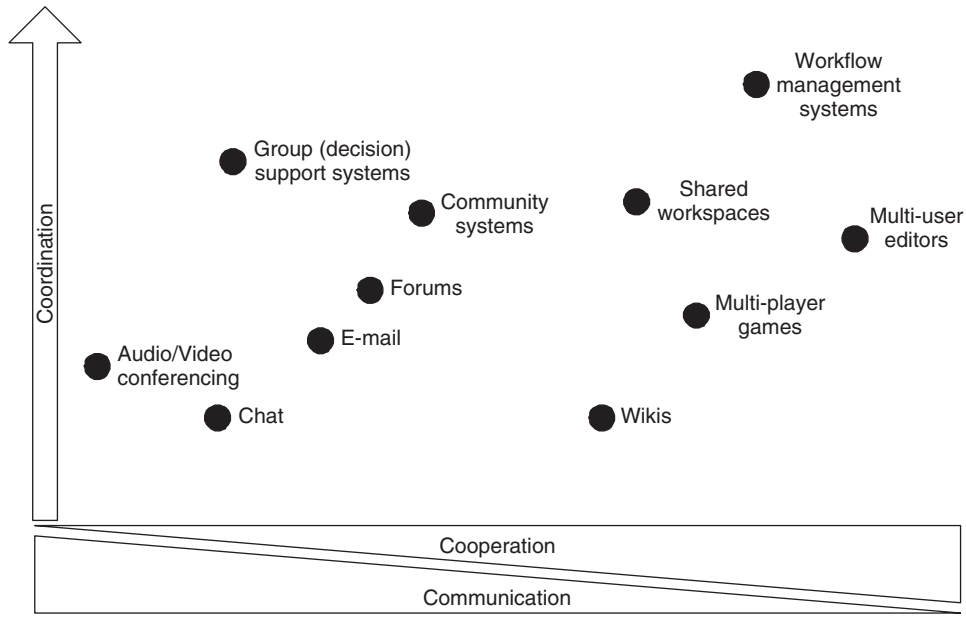

Figure 1.1 Groupware applications in relation to their support of communication, coordination, and cooperation

space. The vertical axis denotes the application's support for coordination, while the horizontal axis is used to denote the degree of communication and cooperation that an application supports. This is possible because a higher degree of communication implies a lower degree of cooperation. By placing an application in this two-dimensional space, the individual degree of communication, coordination, and cooperation can be visualized much better for each of the application types.

In particular, we distinguish the following groupware applications in Figure 1.1:

— *Audio/video conferencing* tools allow users to communicate by various means, so they have a high degree of communication and a low degree of cooperation. Compared to workflow management systems, for example, they do not

explicitly offer functionality for scheduling or organizing tasks. We thus see them at a medium degree of coordination.

— *Chat* tools have a lower degree of communication than audio/video conferencing tools, as non-verbal information is omitted when communicating via a chat application. For the same reason, the degree of coordination is reduced.

— *Group Decision Support Systems (GDSS)* are explicitly designed to support groups in decision-making. For that purpose, they offer synchronous as well as asynchronous communication tools, group votes, etc. They therefore have a high degree of communication and coordination.

— *E-mail* is the most popular groupware application. E-mail can be used for many purposes, but its main purpose is to support communication. As the communication is asynchronous and text-based, the degree of communication is reduced compared to chat tools. However, as users can structure their information when using e-mails, the degree of coordination is increased.

— *Forums* allow users to discuss a topic in which they are interested. The communicating group is therefore defined by the topic. Compared to e-mail, communication is more public. However, if used in a company, forums allow coordination of a group that is cooperating on a common task.

— *Community Systems* integrate a variety of tools and allow a large group of users, i.e. a community, to communicate, to share information, or to coordinate common activities. Often, these tools are web-based. Compared to the tools listed above, community systems have better support for accomplishing and coordinating group goals. However, the degree of communication possible is lower, as there is no possibility of communicating directly with individual community members.

— *Wikis* are web-based systems that allow users to change the content of the web pages. Wikis have their origin in the design patterns community. The first Wiki was the *Portland Pattern Repository*, which was created in 1995 by Ward Cunningham. As Wikis allow users to create and share content, they have a high degree of cooperation, but as they do not explicitly support communication or coordination, they are low in these respects.

— *Shared workspaces* such as BCSW (see Section 1.1) allow users to share content. In the most cases, they also allow structuring of the shared content to coordinate common tasks. For that reason, shared workspaces have a higher degree of cooperation and coordination than Wikis.

— *Multi-player games* are becoming more and more popular. They allow users to solve tasks or quests jointly, and support a number of coordination functionalities for that purpose. Communication is mainly short and used only for coordination, which explains the degree of communication, coordination, and cooperation they exhibit.

— *Workflow Management Systems (WfMS)* are tools that allow modeling, coordination, supervision, and evaluation of a workflow by a cooperating team. For that reason they exhibit the highest degree of coordination of all tools. As their main purpose is to coordinate users in accomplishing a group goal, they have also a high degree of cooperation. WfMS only use communication for coordination purposes, for example to pass on a task or to notify about a completed task, so they show a quite low degree of communication.

— *Multiuser editors* such as CoWord (see Section 1.2) allow cooperating users to create a shared artifact synchronously, for example a text document, drawing, or a spreadsheet, and thus accomplish group goals. This explains the high degree of cooperation of such tools. Multiuser editors use a lot of coordination functionalities as well, for example to avoid conflicting changes. Communication is not explicitly supported, thus the degree of communication is low.

Apart from the various main functionalities that are supported by a groupware application, *awareness* plays an important role. Of the tools listed above, multiuser editors, for example, make use of awareness widgets that show the working area of other users, with the goal to avoid conflicting changes in a shared artifact. Awareness can be seen as a mediator between these three main functionalities.

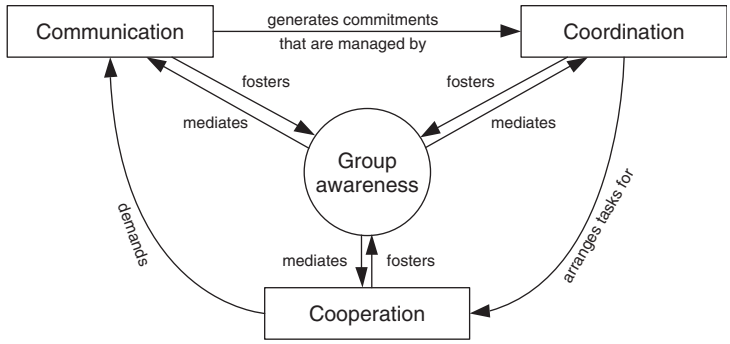

Figure 1.2 Relationship between communication, coordination, cooperation, and mediating group awareness

Gerosa et al. (2004) describe this as shown in Figure 1.2. In this figure, cooperating users must be able to communicate and to coordinate themselves. When communicating, users might generate commitments and define tasks that must be completed to accomplish the common group goal. These tasks must be coordinated so that they are accomplished in the correct order and at the correct time with respect to possible external restrictions. To accomplish these tasks the users have

to cooperate in a shared environment. However, while cooperating, unexpected situations might emerge that demand new communication. In such communication new commitments and tasks might be defined, which again must be coordinated to be accomplished in cooperation. Apart from this cyclic aspect of cooperation, Gerosa et al. place awareness in a central position in Figure 1.2. Every user action that is performed during communication, coordination, or cooperation generates information. Some of this information involves two or even more users, and should be made available to all cooperating users so that they can become aware of each other. This helps to mediate further communication, coordination, and cooperation. Based on this information, users are able to build up a shared understanding of their common group goals and to synchronize their cooperation.

Now that we have clarified our understanding of groupware, the following section presents a scenario in the not too far distant future. This will serve as a running scenario throughout the book. It will relate the patterns in the book to a practical example and show how they can be applied in the scenario.

1.2 A Day with Paul Smith

Join us on a ride with a time machine. Our destination is a typical working day in the life of Paul Smith. Paul is a software engineer and works in the software development department of a leading entertainment device company in London. Currently, Paul is the project leader in the COGE project in which a ***Co**operative **G**ame **E**ngine* is being developed.

Paul's company has subsidiaries all over the world. The members of Paul's team are distributed as shown in Figure 1.3. One team of developers is located in Rio de Janeiro, one in London, and a third in Hong Kong. The main customer is a large game manufacturer located in Germany, which has the goal of building an educational game that helps better understanding of water supply in African countries. The game manufacturer has a group of African pilot users located in Ethiopia and Malawi.

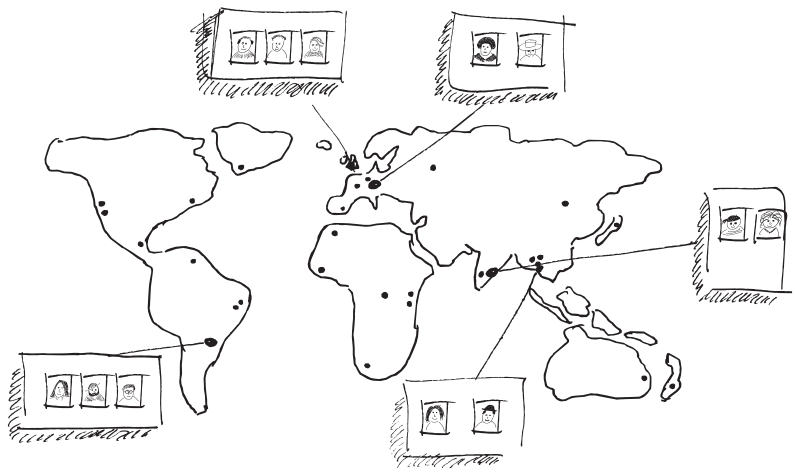

Figure 1.3 Distribution of the team members in Paul's project

Most of the projects in Paul's department are performed in teams to benefit from the synergy of people with varied expertise. Currently, the following interaction constellations are present in the COGE project: the developers from London continue to work on the results that were created only hours before in Hong Kong. Both software teams communicate to plan the internal architecture of the game engine. In other meetings, the London team collaborates with the German customer, which integrates the game engine in its project. The German customer also communicates with some of the developers in Hong Kong or Rio if time shifts allow interaction.

Finally, the German customer interacts with their pilot users and collects suggestions from them on how the game could be improved.

For their common tasks, team members interact daily using their computing devices. Let's now take a look how Paul's typical working day starts.

6:30 AM. The alarm-clock rings and Paul gets out of bed. After a shower and a shave Paul prepares his breakfast. While eating his cereal and enjoying his freshly brewed coffee, Paul has a look at his electronic newspaper (see Figure 1.4). The electronic newspaper is connected to the Internet. According to the preferences Paul has configured, the electronic newspaper shows Paul the latest news in specific categories in which he is interested. Paul is an enthusiastic member of the pattern community and participates in a online community that writes, discusses, and shepherds patterns. He has therefore configured a special section in his electronic newspaper that shows him the latest pattern community news and information about his buddies. The daily report tells Paul what has happened in his online community and allows him to keep track of interesting discussions. A sidebar in the newspaper shows Paul's buddy list. As some of Paul's buddies are already awake, Paul has a short chat with them and agrees to arrange a meeting in the evening.

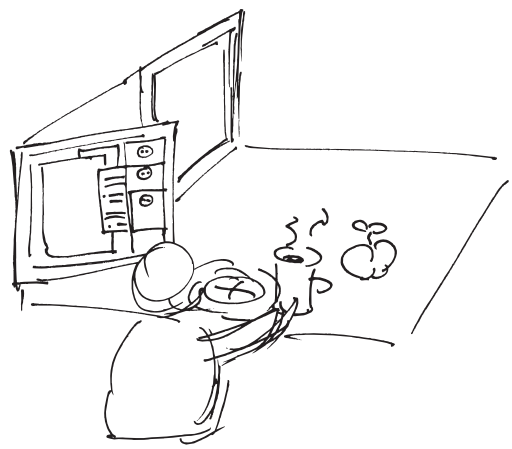

Figure 1.4 Paul has his breakfast

To plan his working day, Paul checks his main tasks for the day and the achievements of his colleagues during the night. In Hong Kong they have solved one of the major problems with the network protocol for the new cooperative game engine. However, the solution has raised some new problems in a module that is developed by the team in Rio. Paul therefore decides to announce a meeting with the colleagues in Rio for the afternoon. He enters the collaboration space and sends invitations to those involved.

8:30 AM. Paul leaves his house in a small neighborhood in the London suburbs, gets into his car, and sets off for his office in the city. In the car Paul recalls the destination from his favorite destinations folder. The navigation system of the car not only connects to GPS satellites, but also to the Internet to plan the best route into the city. It uses GPS to detect Paul's position and the Internet to avoid traffic jams. Additionally, Paul sends his route to his office in an online travel portal that mediates travel mates. Travel mates are selected not only according the destination but also according to Paul's topics of interest. The latter is quite important for Paul, as he does not want to share a ride with someone with whom he has nothing to talk about. In most cases, this allows Paul to pick up a travel mate on his way into the city.

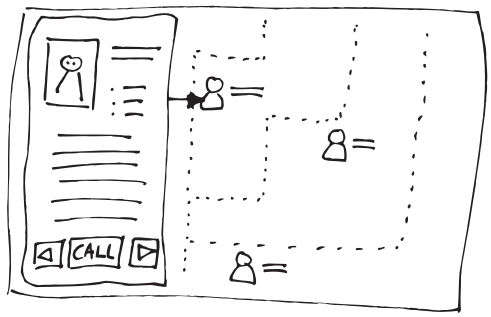

Figure 1.5 Paul looks for a new travel mate

This morning the travel portal suggests a new travel mate (see Figure 1.5). Paul does not know this person, but the portal uses a recommendation system, and the travel mate is ranked as a trustworthy and interesting person. Paul has an additional look at the user gallery and reads the introduction of the proposed travel mate. Paul is satisfied with the suggestion and decides to stop on his way into the city and to pick up the suggested travel mate. The car navigation system calculates the estimated pick-up time and notifies the travel mate. It also keeps the travel mate aware of probable changes so that she does not have to wait too long.

9:30 AM. After picking up the travel mate and dropping her at her destination, Paul arrives in his office. A biometric security check at the entrance proves Paul's identity, and Paul moves to the project's group room where he meets his colleagues. Video screens show the offices of colleagues in Frankfurt, Hong Kong, and Rio de Janeiro in a permanent video stream. One of the colleagues in London starts a discussion about the project's current problems. Paul suggests postponing the discussion until the afternoon when colleagues from Brazil will also be available. Currently, nobody is in the office in Rio, as it is not yet morning there.

As plenty of time is left before the general meeting, Paul joins a group that discusses the software architecture of the current project (see Figure 1.6). This group meets in

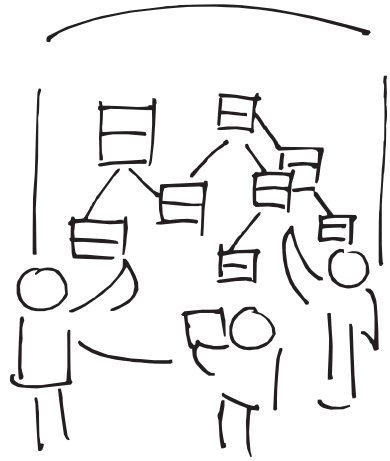

Figure 1.6 Paul participates in a virtual reality conference about the software architecture of the current project

a special room that allows 3D projections. Currently, the group is discussing parts of the architecture for the user interface. Luckily, this group is not affected by the problem that was raised by the solution from China, and makes good progress.

When the meeting about the software architecture is over, Paul goes to his office to start up his desktop computer. He enters the group's collaboration space and is pleased to see that everyone has accepted his invitation to discuss the new problems with the network protocol. The collaboration space also notifies Paul about newly received mails, who else is on line in the collaboration space, and open tasks. As the group has decided to use an open awareness concept, Paul can also see what everyone is currently doing by moving his mouse cursor over the images in his buddy list. This information is often used to start a spontaneous collaboration and discussion about ongoing problems. However, teammates who do not want to be disturbed indicate this in the buddy list so that the collaboration space does not allow direct communication.

1:00 PM. After a few more hours of work and a good lunch in the company's canteen, it is time for the group meeting to discuss the new problems with the network protocol. The video screens show that the necessary people are available at all locations. Paul contacts them and announces the start of the meeting, and all his colleagues move to the group meeting room. This room is equipped with the 3D projector Paul used in the morning. This projector displays video streams for each participant from the various locations (see Figure 1.7), the virtual room for the meeting in the team's collaboration space, and the current shared documents containing the description of the network protocol. This allows everyone to see each other and the material for discussion.

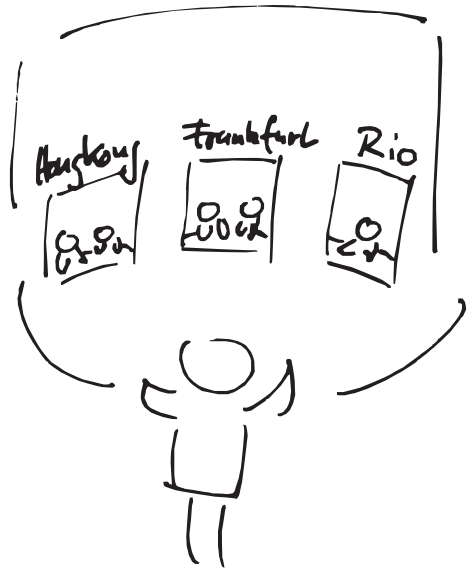

Figure 1.7 Paul participates in a conference to discuss problems with his colleagues overseas

Paul opens the meeting by passing the floor to his colleague Gwan in Hong Kong. Gwan explains how they have solved one of the major problems with the network protocol. To do this, Gwan uses a virtual pointer that allows him to point to the corresponding lines in the source code. Other colleagues can discuss Gwan's presentation using synchronous textual chat so that Gwan is not disturbed. They can also annotate the source code and post questions to a blackboard that will be discussed after Gwan's explanation.

Everyone is impressed with Gwan's presentation, although they know that his solution raises a new problem. After the open questions have been answered, Paul hands the moderation over to Rio de Janeiro and Ana explains the new problem. Ana's presentation raises a lot of open questions on the shared blackboard. The group clusters the open questions and splits into subgroups to address these question clusters. The subgroups create new virtual rooms in the team's collaboration space to discuss the open questions. Before the groups retreat to their new virtual rooms, Paul schedules a new meeting for tomorrow for the groups to present their results.

4:00 PM. Paul has his last meeting for the day. David, a colleague from Detroit, visits the lab. After giving David a short guided tour of the office, Paul tells him about the new problems with the network protocol. David starts smiling, as he knows how to solve part of the problem. Paul and David therefore enter the collaboration space and knock at the virtual door of the subgroup that formed this afternoon

and whose questions David can answer. David offers himself as a mentor and to explain the technology that can solve part of the problem. Soon, David and the other colleagues are in deep discussion and Paul leaves to do other work in his office. Two hours later, David leaves the lab to catch his flight back to Detroit. The subgroup tells Paul that they have nominated David as an expert for specific topics in the collaboration space. This might help David with his next evaluation and wage bargaining.

8:00 PM. Paul has finally finished his most important tasks for the day. He uses his MDA[1] to connect to his online community. As soon as he is on line his friends contact him. They had thought that Paul had forgotten about their appointment. Paul had, and excuses himself for being late. Paul's friends suggest watching a movie in one of the new cinemas downtown. A quick vote shows that all agree. They run a recommender system for movies, and after a short discussion agree on what to see (see Figure 1.8). Adriana offers to buy the tickets and reserve the seats in the cinema's online booking system.

So a long working day finally ends, and Paul leaves his office to watch a movie with his friends. We can step back into our time machine and go on a short ride back to the present.

Figure 1.8 Paul uses the recommender system

[1]MDA is an abbreviation for *mobile digital assistant* which is a combination of a mobile phone and a personal digital assistant (PDA).

1.3 Outline

The scenario of Paul Smith shows one vision of the future. Our main prediction is that in future people will interact more and more using computing devices. In combination with software these computing devices will mediate interaction among people.

As the overview of groupware approaches shows, the scenario is not too far in the future, as most of the computer-mediated interaction it describes already happens in our lives, although not as an integral part of daily life. To mention a few, Paul's day starts with a look at the PERIODIC REPORT of his favorite online community, then at his BUDDY LIST to see who else is already on line. The team is using a collaboration space that is based on virtual ROOMS, Paul's colleague David acts as a MENTOR, and finally Paul and his friends use a recommender system with LETTERS OF RECOMMENDATION to select a movie for the evening.

The terms set in SMALL CAPS are patterns that are part of our pattern language for computer-mediated interaction. These and other patterns can be found in different chapters of this book, which is structured as follows:

— Chapter 2 *From Patterns to a Pattern-oriented Development Process* introduces the reader to the theory of patterns. It looks at the original and more recent publications by Christopher Alexander. Using an end-user centered view, we transfer ideas to the domain of computer-mediated interaction. This results in a pattern form that is different than the pattern forms used in more technical pattern languages. While technical pattern languages use design diagrams or code fragments to illustrate solutions, we prefer a narrative way of presenting the patterns. This ensures that both end users *and* developers will be able to read the solution.

  In the remaining part of this chapter, we will introduce OSDP, a pattern-oriented process for groupware development, which is based on piecemeal growth via short design and development iterations, as well as frequent diagnosis and reflection on working practices and how the application supports them.

— Chapter 3 *Community Support* describes patterns at a high level of abstraction. The patterns in this chapter describe group processes and the use of computer technology to support such processes. Its main focus lies on the early phases of the group process. It answers questions such as:

  - How to arrive in the community
  - How to find out what is interesting in the community
  - How to protect users

— Chapter 4 *Group Support* provides patterns at the user interface level of a collaborative application. The patterns are both technical (describing how to

design group interfaces) and social (elaborating on successful application of groupware technology). Problems solved are:

- How to modify shared material together
- How to shape places for collaboration
- How to organize textual communication
- How to become aware of other user's actions
- How to notice absent participants

— Chapter 5 *Base Technology* discusses the technical layer of groupware applications. The patterns are mainly technical and answer the questions:

- How systems bootstrap for collaboration
- How systems manage shared data
- How systems ensure data consistency

— Chapter 6 *Examples of Applying the Pattern Language* presents two case studies, one on BCSW and another on CoWord. These case studies show how group interaction can be supported by HCHI technology. The goal of this chapter is to put the patterns together and to illustrate how they are used by two well-known groupware applications.

1.4 Acknowledgments

The patterns in this book have evolved over the last five years with the help of many brilliant practitioners in the areas of groupware development, human computer interaction, software development, and social practice design. We would like to express our deepest thanks to all of them and hope that we have not forgotten too many of the colleagues who crossed our paths in the last years.

When we made our first steps towards a groupware pattern language, we had discussions with Alejandro Fernandez, Torsten Holmer (who also contributed the pictures of Paul in the introduction), Jessica Rubart, and Robert Slagter. These discussions were the seed for the idea of a comprehensive groupware pattern language, although we did not anticipate that we would finally manage to write all the patterns in these days.

Our first attempt to write patterns was made in the publication of three initial awareness patterns that were discussed at EuroPLoP 2002. When we look back at the results of this attempt, those patterns look rather naïve now. However, getting in touch with the pattern culture of the EuroPLoP community was probably the most important step towards this book. In endless discussions with shepherds and other authors in writer's workshops we learned to express our patterns in a hopefully clear way that is useful to the book's audience. We are indebted to the EuroPLoP community for being both open to novice pattern authors and at the same time showing a high level of professionalism in the craft of pattern writing.

The following people invested many hours of their time by acting as shepherds for patterns in this book and providing numerous suggestions for improvement: Antonio Rito Silva (EuroPLoP 2002), Kristian Elof Soerensen (EuroPLoP 2003), Joseph Bergin (EuroPLoP 2004), Andreas Rüping (EuroPLoP 2004), Uwe Zdun (CHI2004), Alan O'Callaghan (EuroPLoP 2005), Lise B. Hvatum (EuroPLoP 2005), Ofra Homsky (EuroPLoP 2006), and Munawar Hafiz (PLoP 2006). Uwe Zdun in addition helped us as a shepherd for the whole book. His critical comments forced us to concentrate on the core patterns of the language and bring them to the shape you see today.

Almost seventy pattern authors commented our patterns in writer's workshops at EuroPLoP and PLoP, as well as in pattern workshops at the CSCW conference, the European CSCW conference (ECSCW), and the CHI conference. They are, in alphabetic order: Juan I. Asensio, Paris Avgeriou, Pippin Barr, Joseph Bergin, Bettina Biel, Diethelm Bienhaus, Jan Borchers, Andrea Botero Cabrera, Lynwood Brown, Mishka Bugajska, Frank Buschmann, Jens Coldeway, Andy Crabtree, Catalina Danis, Jutta Eckstein, Amr Elssamadisy, Tom Erickson, Karl Flieder, Richard Gabriel, Ian Graham, Sharon Greene, Tom Gross, Liz Guy, Darren Hayes, Fabian Hermann, Thomas Herrmann, Rod Holland, Torsten Holmer, Stefan Holtel, Ofra Homsky, Lise Hvatum, Mads Ingstrup, Jim Kile, Daniel Kluender, Kari-Hans Kommonen, Gabriele Kunau, Greg Laudemen, Jouni Linkola, Donald Little, Rui Lopes, Michael

Lyons, Mary Lynn Manns, Mika Myller, Mark Prince, Amir Raveh, Simos Retalis, Rebecca Rikner, Judy Roell, Andreas Rüping, Andy Schneider, Dirk Schnelle, Wolfram Schobert, Didi Schütz, Helen Sharp, Marianna Sipos, Guy Steele, Winfried Tautges, Lucia Terrenghi, John C. Thomas, Markus Völter, Aake Walldius, Charles Weir, Michael Weiss, Leon Welicki, Dave West, Elizabeth Whitworth, Michael Wissen, Mary Zajicek, Uwe Zdun, and Jürgen Ziegler.

Several colleagues in the research field of Computer-Supported Collaborative Work (CSCW) provided us with interesting insights into research prototypes, participated in discussions on the nature of collaborative applications, or helped us to find known uses for the patterns in this book. To name only the most important contributors, we would like to thank Christian Schuckmann, Jan Schümmer, Holger Kleinsorgen, Hans Scholz, Andrea Kienle, Peter Tandler, Michael Koch, and Carl Gutwin. Special thanks are due to Elke Hinrichs (Fraunhofer FIT), Wolfgang Graether (Fraunhofer FIT), and Thomas Koch (OrbiTeam) for giving us a guided tour through the visual and invisible parts of BSCW. Their input greatly helped to document the BSCW case study. Similarly, special thanks are due to Chengzheng Sun and Steven Xia, of the School of Computer Engineering, Nanyang Technological University, Singapore, for their help and input for the CoWord case study.

Jörg and Anja Haake encouraged us to use the book in our classes on designing collaborative systems. We hope that many classes of future students will benefit from this decision. In this context, we would also like to thank our students, who started using the pattern language from the winter term 2004. Observing their use of the pattern language to design collaborative applications helped us to see the language from a student's point of view.

We thank all the people who have actively participated in the production of the book. These are the series editor Frank Buschmann, Gaynor Redvers-Mutton, Sally Tickner, Rosie Kemp, Andrew Kennerley, Hannah Clement, and everyone else at John Wiley & Sons. Special thanks are due to our copy-editor Steve Rickaby, who showed patience even on a very tight schedule. It was a pleasure to discuss the nuances of the English language with Steve.

Above all, we would like to express our gratitude to our families. Without their support, a book like this would not have been possible.

And finally, we thank you for your interest in this book. We hope that our pattern language will prove useful to you and become a tool in your daily work when shaping better computer-mediated interaction.