TECHNISCHE UNIVERSITÄT MÜNCHEN

INSTITUT FÜR INFORMATIK

A Software Architecture for Knowledge Acquisition and Retrieval for Global Software Development Teams

Andreas Braun

INSTITUT FÜR INFORMATIK
DER TECHNISCHEN
UNIVERSITÄT MÜNCHEN

Forschungs- und Lehreinheit I
Angewandte Softwaretechnik

A Software Architecture for Knowledge Acquisition and Retrieval for Global Software Development Teams

Andreas Braun

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. Helmut Krcmar
Prüfer der Dissertation:	Univ.-Prof. Dr. Johann Schlichter
	Univ.-Prof. Bernd Brügge, Ph.D.

Die Dissertation wurde am 24.03.2004 bei de Technischen Universität München eingereicht und durch die Fakultät für Informatik am 01.07.2004 angenommen.

– Kurzfassung –

Software-Projekte erfordern intensive Kommunikation sowie die Zusammenarbeit verschiedener Parteien und Vertreter von Interessensgruppen, die jeweils unterschiedliche Positionen einnehmen. Ein Großteil dieser Kommunikation ist informell und dient z.B. der Diskussion, der Klärung von Fragen, Verhandlung, bzw. dem Aufdecken von Wissen. Projektbezogene Kommunikation wird zunehmend vielschichtig durch den kombinierten Einsatz elektronischer und nicht elektronischer Kommunikationsmittel. Dies gilt sowohl für lokal an einem Ort durchgeführte Projekte, als auch – in besonderem Maße – für verteilte oder gar globale Software-Projekte. Kommunikationsfehler sind in diesem Zusammenhang kostspielig und gefährden den Projekterfolg, insbesondere wenn sie in den frühen Phasen des Lebenszyklus auftreten. Aus diesem Grund ist die wissensbasierte Kommunikations- und Kooperationsunterstützung aller Phasen des Projektes erforderlich.

Ausgehend von Studien, die auf grossen und verteilten Projekten durchgeführt wurden, darunter einer Studie des Autors, haben wir iBistro entworfen. iBistro ist eine experimentelle Umgebung zur Erfassung von Wissen in den verschiedenen Phasen globaler Software-Projekte. Dabei werden die gesammelten Informationen in einem *Rationale*-basierten Ansatz strukturiert. Das so gesammelte Wissen wird den verschiedenen Projektbeteiligten in späteren Projektphasen oder neuen Projekten durch das System zur Verfügung gestellt. Dabei ist das Ziel von iBistro, die bestehenden Prozesse und Methodiken einer Organisation so wenig wie möglich zu beeinflussen und dennoch sowohl Umfang und Qualität des erfassten Wissens zu verbessern. iBistro sammelt somit Informationen durch eine Vielzahl von Wissensquellen (*Knowledge Sources*) und stellt dieses Wissen über anpassbare Wissens-Sichten (*Knowledge Views*) wieder zur Verfügung. Das somit aufgebaute Wissens- und Informationsnetz bildet ein elektronisches Gruppengedächtnis (*Group Memory*).

Diese Dissertation stellt eine gemeinsame Architektur für Wissenserfassung, Wissensaquise und Wissensverwendung von Projektwissen vor. Darüber hinaus bezieht sich der Begriff „Projektwissen" auch auf Ergebnisse der Arbeitsprozesse, wie zum Beispiel Dokumente oder Quellcode. Dies führt zu komplexen Querverbindungen zwischen solchen Artifakten und allgemeinen Wissens-Informationen. Die Architektur, genannt *Distributed Concurrent Blackboard Architecture* (DCBA), erweitert das aus der Künstlichen Intelligenz bekannte Konzept der Blackboard-Architektur. Die DCBA-Architektur ermöglicht die nebenläufige Erfassung von informellen Informationen und Wissen in vielschichtigen Kommunikationsmitteln und -wegen, wie z.B. Arbeitstreffen, in Brainstorming-Treffen, Email, oder bei der Verwendung von Entwicklungswerkzeugen, wie z.B. der Versionskontrolle. Die DCBA-Architektur wurde dabei speziell für die Unterstützung verteilter Teams in synchronen und asynchronen Szenarien entwickelt.

In einer Fallstudie wurde die Implementierung der DCBA-Architektur zwischen der National University of Singapore und der TU München getestet und iterativ weiterentwickelt. Dabei deckte das verteilte Szenario die Schwierigkeiten bei der verteilten Projektarbeit mit unstrukturierten und nicht integrierten Werkzeugen im Vergleich zur Verwendung der DCBA als Wissensquelle für eine kleine verteilte Arbeitsgruppe auf. Diese Dissertation schlägt eine spezielle Team-Formation für Globale Software Entwicklung, das *balancierte Team*, vor und diskutiert deren Unterstützung in iBistro.

A Software Architecture for Knowledge Acquisition and Retrieval for Global Software Teams

"The soul is the same in all living creatures, although the body is different." [HIPPOCRATES]

andreas.braun@accenture.com
Pfefferstraße 3
82194 Gröbenzell

Version of
29th July 2004

– Abstract –

Global software development is communication intensive. It requires the collaboration of many stakeholders representing different positions. The stages of the development life cycle are characterized by many informal interactions, often face-to-face, whose aim is to clarify, brainstorm, negotiate, or recover knowledge. Project-related communication is also becoming more and more diverse, especially if distributed globally. It includes various electronic and non-electronic means of communication, both for on-site and off-site development. Moreover, project knowledge refers to artifacts of the development life cycle, such as source code or documents. This results in complex cross references between artifacts and knowledge items. Misunderstandings and communication mistakes are costly, especially those introduced early in the life cycle. Hence, it is essential to support communication and cooperation during the overall life cycle.

Based on the results of studies in distributed and large projects, including one performed by the author, we have designed iBistro, an experimental environment for capturing knowledge during the various stages of both distributed and single site software projects, structuring the knowledge during a rationale-based approach, and retrieving the knowledge during subsequent stages of the development process or later projects in the same organization. In developing iBistro, our goal is to interfere as little as possible with the development methodology and tools of an organization, while improving the coverage and the quality of the knowledge captured. Thus, iBistro collects information using a variety of knowledge sources and provides customizable knowledge views to build up what we call *group memory*.

This research further proposes a common architecture for knowledge capture, acquisition, and retrieval. This architecture, called the *distributed concurrent blackboard architecture* (DCBA), extends the concept of a blackboard architecture in artificial intelligence. The DCBA enables the concurrent capture of information and knowledge through many communication vehicles, such as meetings or brainstorming sessions (using i.e., SMART Boards), email, or artifact-based tools (e.g., version control). The DCBA in its distributed nature is designed specifically to support distributed balanced teams in synchronous and especially asynchronous settings.

An implementation of the DCBA has been developed and tested in a distributed setting between the National University of Singapore and Technische Universität München. The distributed scenario revealed the difficulties in distributed work introduced by unstructured and non-integrated communication tools in contrast to the DCBA as a knowledge repository for the small workgroup. The dissertation suggests a team-formation to deal with global software engineering, the *balanced team*, and how they can be supported by the DCBA.

Keywords: **Technologies & tools for distributed development; Communication, collaboration, and knowledge management in distributed organizations**

ACKNOWLEDGMENTS

Pursuing this thesis made clear to me that this would have never been possible without the support and encouragement of a lot of people.

First, I would like to thank my counselor at Accenture and founder of this research, Frank Mang. Frank has been my mentor, my confidant, my colleague, and a fount of moral support in pursuing this dissertation and balance it with the challenges of project work at Accenture.

At Accenture, I found an open and interested atmosphere and much support right from the start. Martin Illsley, Ph.D., at the Accenture Technology Labs (formerly C-Star), Sophia Antipolis, France, supported me in my ambition to visit researchers at Accenture Technology Labs and discuss my research proposal and current areas of research at Accenture. Edy Liongosari and Igor Gordon at Accenture Technology Labs in Northbrook, Il., and Luis Monterro, 161 North Clark office, Chicago, hosted me and introduced me to recent Accenture research (and by the way made possible an interesting stay at Chicago). Wolfgang Behr supported me in designing and especially performing the communication study.

I also want to thank the fellow employees, colleagues, and doctoral students at the chair for Applied Software Engineering and specifically the members of the Global Software Engineering (GSE) research group at TU-München. The informal meetings in the GSE group helped to clarify a lot and also enabled a larger research program, including related efforts and research in the GSE group.

I am also grateful for the industry contributions I received. I have to thank SMART Technologies Inc., Toronto, Canada, for accepting my research proposal and supporting the chair with two SMART Boards^{TM}.

Many research projects at university would not be successful without the support of students writing diploma, master's, or bachelor thesises. In my case, I have to thank Oliver Hengstenberg, Lilith Al-Jadiri, Roland Zumkeller, Guying Hu, and Florian Nikitsch.

Especially the students in longer-term appointments made important contributions to iBistro. Oliver Hengstenberg implemented and tested the MEETINGGENERATOR tool during his diploma thesis "Video-based Capture of Rationale Information in Informal Meetings". Roland Zumkeller started with evaluation and installation of the SMART Boards at the chair and ported the MEETINGGENERATOR to the SMART Board. Roland also started with implementing and evaluating the knowledge taxonomy for the repository and designed an initial version of the database.

Guying Hu implemented the first prototype of iBistro's architecture during his bachelor thesis. Guying also supported the case study between Singapore and Munich and participated in several "global debugging sessions" – often in the early morning at Munich. Florian Nikitsch jumped right into the project with many hours of bug-fixing. Roland also worked from Singapore in the globally distributed case study. Roland finally implemented and tested the distributed concurrent blackboard architecture. Florian also investigated the performance of the iBistro system in this global setting and made important changes to improve iBistro's speed.

I want to acknowledge my supervisor and principal reviewer, Prof. Bernd Brügge, Ph.D., for his guidance and comments on this dissertation and the publications that were written while pursuing my dissertation.

I also want to thank my second reviewer, Prof. Dr. Johann Schlichter, for his comments on related work during the earlier stages of the dissertation and for his fast review and valuable comments on the final draft.

Finally, I want to thank Allen Dutoit, Ph.D. for his guidance, support, and coaching during three and a half years of dissertation and for helping to negotiate a suitable and contributional research setting between TU-München and Accenture.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

“Software is Hard.”

[DONALD E. KNUTH],
Oct. 5th, 2001, at Technische Universität München.

We believe that globalization and distribution of companies and teams is a matter of fact for the future work environment. Just as one would diversify the risk in a stock portfolio companies need to diversify geographic locations. With advances in technology, no longer is it essential for employees to be located in one location. Many companies are reevaluating their location strategy and deciding to diversify their risk, because the permanent availability of company-resources, such as IT infrastructure, is vitally important for most organizations. Investigations in the banking industry showed, that a complete breakdown of IT infrastructure for two days would likely ruin an international banking corporation [Kaufman et al., 2002].

In the short term many companies simply distribute operations and employees in different buildings in different parts of the city or in nearby locations. However in the long term, technology advances such as video conferencing, video streaming and local broadband make distributed work look like local work again and hence enable the virtual workplace. There should be little or no limitations on where business operations and employees can be located.

The experiences and problems with software labs with students at Technische Universität München, Carnegie Mellon University, and National University of Singapore inspired our desire for an “intelligent workplace” for all locations the student classes would take place at. This dissertation describes the software architecture and specific tool support for *iBistro*[1]. iBistro is originally meant to be an informal coffee room to support any type of casual meeting or brainstorming

[1]The term iBistro goes back to the initial idea of an augmented coffee room. The atmosphere of a bistro, so the idea, shall foster the informal nature of meeting in iBistro.

session and therefore help vanquish the differences in time, location, and culture.

During several stages of testing and working with iBistro prototypes, the importance of supporting local teams in their location, while interconnecting such teams in different sites became obvious. iBistro hence allows for *synchronous* support of local team meetings and makes the outcome (minutes) of the local meeting available to *asynchronous* users. (For instance, remote users at a different time or for later use in the same location.)

This dissertation inspects the requirements for iBistro as an exemplary prototype for an intelligent meeting room and describes the realization of five fundamental concepts of iBistro: knowledge capture, knowledge acquisition, storage, knowledge representation, and knowledge retrieval. The main focus of this dissertation is to collect sufficient insights and actual data to allow for the design and implementation of a technical infrastructure – in terms of a framework – for improved project communication and project management in typical software projects, regardless of the chosen approach (i.e., programming language, development methodology, and tool support).

DeMarco and Lister observed 1979 in a survey that fully twenty-five percent of projects that lasted twenty-five man-years of work or more failed to complete. In the early surveys, they discarded these failed data points and analyzed the others. Since then, DeMarco and Lister have been contacting whoever is left of the project staff to find out what went wrong. For the overwhelming majority of the bankrupt projects studied, *there was not a single technological issue to explain the failure.* [DeMarco and Lister, 1979]

In the majority, the surveyed team members named organizational 'politics' as cause for the project to fail. Politics, however, is a rather broad term in the context of software engineering. In many cases, further investigation showed that the real reasons for project failure are to be found among the following problems: communication, staffing, disenchantment with the boss or the client, lack of motivation, or lack of turnover [DeMarco and Lister, 1979].

DeMarco's and Lister's final assessment[2] is: "The major problems of our work [software development or IT consulting] are not so much technological as sociological in nature."

Following DeMarco and Lister, the project sociology, including communication, is the main trigger for project failure. Communication in turn, is responsible for the exponential increase in cost of change [Brooks, 1995]. The resulting negative return on investment (ROI) is an indicator for failure of such a project. Figure 1.1 shows the coherence between cost of change and return on investment leading to project failure. The figure shows three phases. First, on project inception, the cost curve is flat and ROI is positive. Second, the cost starts increasing

[2]Further studies and investigations are discussed in Chapter 4.

with the number of changes. Cost quickly overcome additional value added from the features or change implementation. Third, changes made after the cost become exponential, the additional cost overwhelms all return gathered from the product so far, leading to project failure.

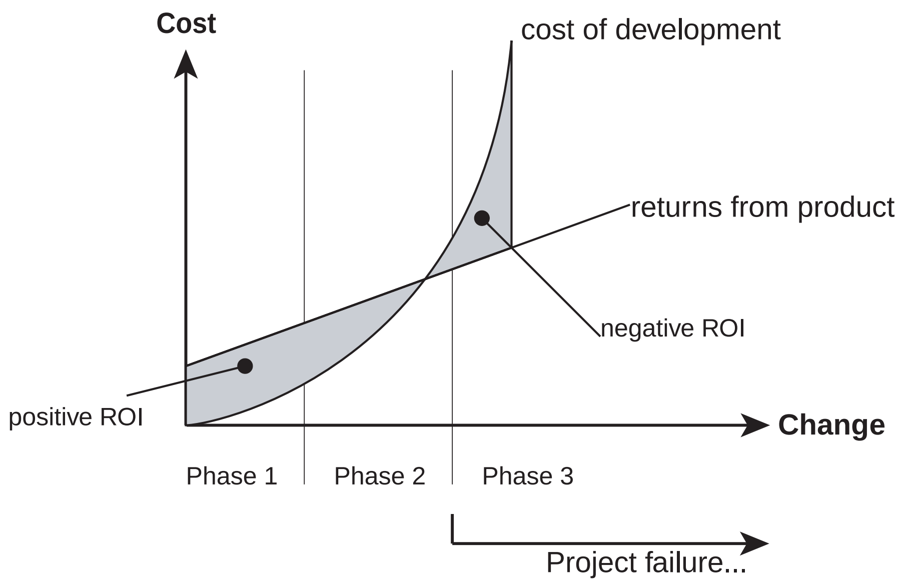


Figure 1.1: Exponential cost of change leading to project failure.

Today, *complete* failure is not the only and might not be the major threat to a project manager.Management understands the importance of teams and teamwork, human aspects in development, and sociology. However, software is still frequently delivered late, over-budget, and with many remaining errors. Additionally, a competitive market places extreme demands on software development managers, teams, and the products they produce. In many cases a barely acceptable product developed quickly will be preferred to an excellent one produced slowly, especially under time to market pressure. Many invisible attributes of a product as well as attributes of the process that created the product are often the most compromised in the name of acquired market share or a better cost to profit ratio. Examples for attributes neglected are the software architecture on the technical side, and design rationale documentation on the process side. Both add overhead to the project that pays off only in a subsequent project, for instance in a new release or a new project.

Software developers have to deal with three categories of problems: First, “soft” issues such as people matters and related issues in communication, human performance, and work organization; Second, essential and technical problems of deciding *what* to develop; Third, accidental and methodical problems of deciding *how* to develop (for instance, how to model it, what tools to use, what operating system to install).

The “soft” issues of software development are inextricably bound to the structures, processes, tools, and outcomes of the organization. By building and understanding these elements, project managers can engineer them to enable themselves and their developers to create successful systems. The categories of issues in software engineering – knowledge, communication, technology, and management – are interlocked. For instance, an essential problem in developing a software system is to specify, design, and test the conceptual constructs of data sets, data items, and algorithms. During the early phases of the project, one has to communicate a lot with various stakeholders. The challenge is to save this information for later re-use to start the process of translating requirements into software. During design and implementation, one has to recapitulate information and knowledge captured earlier. And during all the phases in a concrete project, we have to deal with people, essence, and accidents.

The key is to find a way to integrate the essence/ technical issues, the accidental/ metholodical issues, and people in a software development project.

1.1 THESIS CONTRIBUTIONS AND GOALS

This dissertation proposes integrated tool-support to take into account three major issues in software development: First, human communication during the overall process in single-site and globally distributed settings. Second, deciding ‘what” to develop, during the proposal and requirements engineering; Third, the transition from requirements to implementation by recapturing earlier knowledge during design and implementation, including later phases, such as testing or roll- out.

The early phases in a software project, while being crucial for the eventual success, are difficult to structure and much of the content and rationale generated during those phases is often lost. Requirements engineering is communication intensive and requires the collaboration of many stakeholders representing different positions. For example, the early stages of requirements are characterized by many informal interactions, often happening face-to-face. The aim of early requirements meetings is to clarify, brainstorm, and negotiate requirements. While the final requirements describe “what to build”, part of the discussion leading to that stage contain the rationale argumentation; stakeholders in this process, such as clients, domain experts, consultants, or developers, are consolidated as identities; the location a meeting took place in represents the “where” in this model.

iBistro’s group memory is a knowledge base that is designed to answer questions that occur during the distributed software development process, for example to find responsible stakeholders or artifacts such as documents or source code. Many of these issues are answered during requirements elicitation. It is, however, very important to capture those early phases to have the issues available in the later phases of development. For instance to automatically create a list of all issues stakeholders identified during requirements meetings in the design phase. This can also be used to automatically create a list of all open issues.

Because requirements are the hardest part in development [Brooks, 1995], we specifically investigated requirements engineering. Misunderstandings and mistakes introduced during the early phases are costly, as they impact on all other phases of software development[3]. Also, content, knowledge, and rationale that have been lost are hard to reconstruct, if ever possible. Hence, the key is to support and capture communication during requirements engineering. While much research in requirements engineering has been done to support formal negotiation during requirements [Boehm et al., 1998; Damian et al., 2000; Kotonya and Sommerville, 1996; Sutcliffe and Ryan, 1998; Al-Rawas and Easterbrook, 1996], much less attention has been paid to brainstorming and informal meetings [Nakanishi et al., 1999; Dourish, 1996; Braun et al., 2001a].

We describe features of iBistro, an experimental environment for capturing informal meetings with roomware[4] technology, structuring the meeting minutes along with other project-related information using a *rationale-based* approach, and retrieving the minutes during subsequent informal and formal meetings. Along with capture and retrieval of knowledge, iBistro features knowledge acquisition and links knowledge and information from distributed different-location and different-time meetings in shared and distributed knowledge repositories.

In summary, the scope of the dissertation is:

- To support requirements for distributed projects by providing a tool for collaboration support.
- To support informal meetings and brainstorming sessions.
- To realize a ’group memory’ to support distributed balanced global development teams.
- An ontology and taxonomy to allow for information to knowledge transition.

The technical contributions in iBistro are:

- A new software architecture exceeding the capabilities of the original blackboard model. This comprehends a transparent infrastructure for knowledge acquisition and search for answers to development problems.

[3]This issue is also addressed, but with a different approach, by agile and iterative software development [Beck, 1999; Kruchten, 2000].

[4]*Roomware* is a collective term for electronic equipment to equip rooms with in order to improve collaboration and foster team-work.

- An implementation and evaluation of the transition from the distributed concurrent blackboard model to persistent relational data storage.
- Implementation of specific knowledge sources for audio and video capture of informal meetings.
- Implementation and evaluation of a group memory that enables the integration of various communication media.
- Implementation and evaluation of a common framework and architecture to allow for future development of distributed knowledge sources and views for iBistro or similar tool support.

1.2 DISSERTATION OUTLINE

The dissertation is structured in three parts. Part 1 describes the problem domain and identifies requirements for a tool support. Part 2 describes the analysis and design for a visionary iBistro system on the base of scenarios. A software architecture and a framework for iBistro is suggested. Part 3 describes the actual implementation, lessons learned during testing and a case study of a distributed project between TUM and the National University of Singapore.

PART 1: REQUIREMENTS ELICITATION & ANALYSIS

Chapters 2 through 4 serve as input for the requirements and provide requirements specification for the collaborative tool support:

Chapter 2 defines various terms and introduces the solution space. In the chapter, we categorize group work and computer supported cooperative work, human-computer interaction, knowledge management, introduces blackboard software architecture style, and discusses empirical methods in social research and quantitative metrics.

In Chapter 3 we introduce informal meeting support as an application domain in distributed and global software projects and describe typical organizational structures in industrial software projects. We argue that global software projects, given the advancements in software engineering over the past two decades, are a consequence of “division of labor”in non-distributed organizations. We then describe the research context and test beds for the implementation prototypes and tool-support.

In Chapter 4, we take a closer look at the non-technical aspects, in particular the communication vehicles used in team communication and cooperation during a industry software project. We investigate the specific requirements for iBistro as a team support tool by examining intra and extra-team communication during the implementation phase of a large software development project in the banking industry.

The chapter introduces the concept of a "balanced team" to allow for both improved project success as well as distributed development. We compare two teams that were organized as a balanced team against eight teams with a more traditional composition as found in large projects.

We look at the project situation that built up the environment and experimental conditions for our survey. Based on a scientific evaluation using the quantitative and qualitative methods introduced in Chapter 2, we then present and assess the results of our survey and relate them to global software engineering.

PART 2: VISIONARY SCENARIOS AND DESIGN

In Chapter 5, we describe iBistro as a testbed for experimentation with distributed software teams, in particular for support of augmented and informal meeting spaces.

We also discuss related and competing systems to iBistro. In particular, we investigate support systems for (synchronous) face-to-face meetings and live remote conferencing.

Chapter 6 starts with an overview of the technical architecture of the solution, and argues for a blackboard-based approach to support distributed teams of developers. The approach is broken down into five fundamental concepts for the system: information and meeting **capture**, knowledge **acquisition**, knowledge **storage**, knowledge **representation**, and knowledge **retrieval**. These concepts are then mapped to components of the architectureand discussed in detail. Finally, we give an overview of the resulting system and discuss its properties.

PART 3: ITERATIONS, DISCUSSION AND FUTURE DIRECTIONS

Chapter 7 describes the empirical approach of our research. We present the iBistro prototypes and components used during our case study. The iBistro components were developed in several iterations. In the first iteration we designed, developed, and evaluated the MEETINGGENERATOR component. In the second and third iteration, the software architecture, the framework for the distributed concurrent blackboard architecture, and the test harnesses used were designed, implemented, and tested. The chapter also discusses the lessons learned from the research prototype.

In the final chapter we describe expected obstacles and issues that likely would occur if iBistro or the distributed concurrent blackboard architecture is used outside an academic context, for example in the IT consulting industry. We close with an outlook on how this research could be used and continued at the Chair for Applied Software Engineering at the Technische Universität München and specifically within the Global Software Engineering research group at TUM.

The appendix contains a glossary, a catalog of a abbreviations, further implementation details, research material, such as interview guides and statistic material, and finally a bibliography.

1.3 NOTATION

Throughout this dissertation, we will use the following notation and typesetting: *Italic* typeface indicates the introduction of a special keyword or term that will be used consecutively. **Boldface** is used to emphasize important catchwords or results. Sans-serif font in the navigational text refers to names and labels used in a figure or table; the `typesetter` font is used for computer text, such as classes and programming language.

CHAPTER 2

TERMS AND DEFINITIONS

Die meisten Definitionen
sind Konfessionen.

[LUDWIG MARCUSE]

In 1984, Paul Cashman and Irene Grief coined the term "computer-supported cooperative work" (CSCW) during a workshop of people from various disciplines who shared the common interest in how people work and how technology could support them. Since then, researchers have continually worked on electronical support for cooperating people and groups. Even though researchers were interested in group support earlier, CSCW started a new era by integrating new technology, such as minicomputers, and, more importantly, various scientific disciplines [Grudin, 1994].

While CSCW refers to the abstract application domain, groupware usually refers to the corresponding software solutions or systems. We use the following definition for groupware:

> **Definition.** *Groupware* are computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment. [Ellis et al., 1991]

Groupware systems can support both interactive and non-interactive collaboration. Examples for *non-interactive collaboration* are people collaborating via newsgroups, electronic messaging, or shared filesystems that address the *asynchronous* remote collaboration of people. In *interactive collaboration*, participants work together *synchronously*. Examples are meetings, telephone calls, conversations and related technical support, such as whiteboards etc.

The users of a groupware system are interconnected people who work towards a common goal and share common interests. Groupware systems include the notification of users and the awareness of other participants, in particular the awareness of the concurrency of their work.

2.1 USERS OF GROUPWARE SYSTEMS

The users of groupware systems can be teams or communities. In this dissertation, we will consecutively use the following definitions.

If an organization spans social or other borders that delimit communities, it is likely that teams from different *communities* have a working contact. The term community spans two areas: first, actual social groupings and second the particular quality of relationship. A community indicates actual social groups by comprehending people with common interests or living in a particular area. The community describes a state of organized, while traditionally relatively small, society, and the locality (for instance the people of a district). The particular quality of relationship is indicated by the quality of holding something in common (as in community of interests or communities of goods) and the sense of common identity and characteristics.

Local groupings are based on proximity and sometimes on face-to-face relationships (as in a local community or in community work). Communities of interest are for instance research or business communities. Community characteristics occur in ethnic communities. The community has a specific quality of relationships by sharing of similar attributes, values, identities, and by participatory decision. In a community, this is connected with emotional and moral investments.

In this dissertation, we will use the following definition for community:

Definition. A *community* is a group of people who are sharing some similarities, such as language and having a particular quality of social relationship.

If the common similarity of a community is, for instance, common language and social togetherness, community borders are crossed if communication exceeds these properties by addressing external persons belonging to a different social class, i.e. speaking a different first language.

Definition. *Community borders* are the boundaries of a community. If communication exceeds the limits between two specific communities by overcoming the common similarities which define the community, it *crosses* community borders.

If a community works together based on electronic means, it becomes a *virtual community*. A virtual community is a community of people sharing common interests, ideas, and feelings over the Internet or other collaborative networks. Rheingold defines virtual communities as follows:

> **Definition.** A *virtual community* is a social aggregation that emerges from electronic communication (for instance using the Internet) when people carry on public discussions with sufficient human feeling to form webs of personal relationships in the electronic media. [Rheingold, 2000]

In other words, the virtual community is a team formation that spans locality and is interwoven by electronic communication vehicles. Virtual communities existed before the world wide web (WWW) on bulletin board services (BBS). Some virtual communities or facilitators of them use the metaphor of a coffee house to help users visualize the concept. In general, there are two kinds of communication among virtual community members: asynchronous message postings and synchronous chat. Usenet newsgroups are an example of the former. Many Web sites, such as Geocities focus on subject information exchanges. For real-time chat, Internet Relay Chat (IRC) is a system used by many Web sites that foster virtual communities.

The notion of a community can be split in two variants, depending on the objectives that led to the formation of a community. In large organizations, experts of local or virtual teams together can form a *community of interest* sharing some common interest. For instance, the IT security experts for all professional practices may form a worldwide community of interest for the specific domain of IT security, while all software developers interested in software architecture may form an architecture speciality independently of their application domain.

> **Definition.** A grouping of users who generate a majority of their communication traffic in calls to other members of the group is called *community of interest* (also known as *special interest group*). The community of interest may be related to a geographic area or to an administrative organization.

The second variant is the *community of practice*, which is build from the conforming involvement in a common process.

> **Definition.** The *community of practice* shares the joint possession of a common process, still pursuing different professional goals. [Koch, 2003]

The most advanced form of a community is a team [Borghoff and Schlichter, 2000].

Definition. A *team* is a group of interconnected people who work together on a temporary assignment, such as a project. The team works towards a common goal, sharing some artifacts [Borghoff and Schlichter, 2000].

Teams are a omnipresent reality in today's projects and define how people relate to one another in the integration of their task, social, and emotional activities. For instance, a local requirements engineering team in Germany can be in contact and work together with a software development team in Asia.

Professional software development teams in non-distributed scenarios work based on proximity in a sense of working for instance in the same building. They share the same professional interest as they work on the same problem and towards a common goal. Software development teams typically share the same company culture.

eXtreme Programming (XP) [Beck, 1999], for instance, is based on small and *co-located teams*. Co-located teams cover almost all of the affected groups and individuals in the software development process. The three most important success factors for co-located teams are culture, people, and communication [Lindvall et al., 2002]. Agile Methods need cultural support otherwise they will not succeed. Competent team members are crucial. Agile Methods use fewer, but more competent people. Physically co-located teams and pair programming support rapid communication. Close interaction with the customer and frequent customer feedback are critical success factors. Large distributed teams may need to rely on more documentation and adopt RUP [Kruchten, 2000] or other less agile processes.

2.2 DISTRIBUTION

Teams and software developers were organized co-located to improve communication, hence reducing communication errors and finally risk. In distributed scenarios, communication media is needed to interconnect teams and distributed communities. Being virtual communities, *distributed communities of interest* are typically interconnected using electronic means, such as a shared knowledge base or discussion forums. Communication media such as the internet or videoconferencing and the broad availability of affordable and fast travel enable distribution in various professional scenarios, including development of software on two or more different continents.

2.2.1 LEVELS OF DISTRIBUTION

Independently from the geographic location, a meeting can take place at the same time (*synchronous* meeting) or at different time (*asynchronous* meeting). A face-to-face conversation or a phone call are examples of synchronous meetings. Communication can also take place *asynchronously* (different time). Asynchronous communication is important in scenarios where co-workers are located in different time zones.

Grudin [Grudin, 1994] considers the distribution in time and space with respect to mobile communications and extends the *time-place* matrix by aspects of mobile communication. (See Table 2.1.)

Mobility is an important aspect in distributed scenarios. In a virtual community, it is more difficult to predict the location a user is reached in if mobile communication is used. Mobile developers cannot fall back on the full set of resources (such as documents, files, technology, . . .) as a team member in the office could.

	Same time Synchronously	**Different time Asynchronously**	
		Predictable	**Unpredictable**
Same place	Face-to-face meeting	Work shifts	Blackboard, team room
Different place Predictable	Video conferencing, phone call	Email	Collaborative group editing
Different place Unpredictable	Mobile phone conferencing	Non-realtime computer conference	Transaction processing, workflow, b-boards

Table 2.1: Distribution over space, time, including aspects of mobile communication. [Grudin, 1994; Borghoff and Schlichter, 2000]

Users can be reached in an unpredictable location if they are using a mobile phone. A mobile user, for instance, might be reached away from the workplace. As a result, a mobile user could be unable to cooperate with a colleague, for instance due to missing resources or documents. In contrast, the location of a same time/ different place non-mobile phone call, is predictable, as the location of the telephone is known (if the telephone call is not forwarded).

Unpredictable time, on the other hand, results from the latency of the answer. While an answer to a question in an ongoing telephone conversation can be expected immediately, the latency for a email is less predictable.

In addition to the distribution over time and space, globally distributed teams have also to deal with social and community borders. Figure 2.1 shows the distribution over time, space, and community in a 3-dimensional space.

This research started by supporting same place informal meetings, whether they occur synchronously or not and in the same community or not (Cells A and

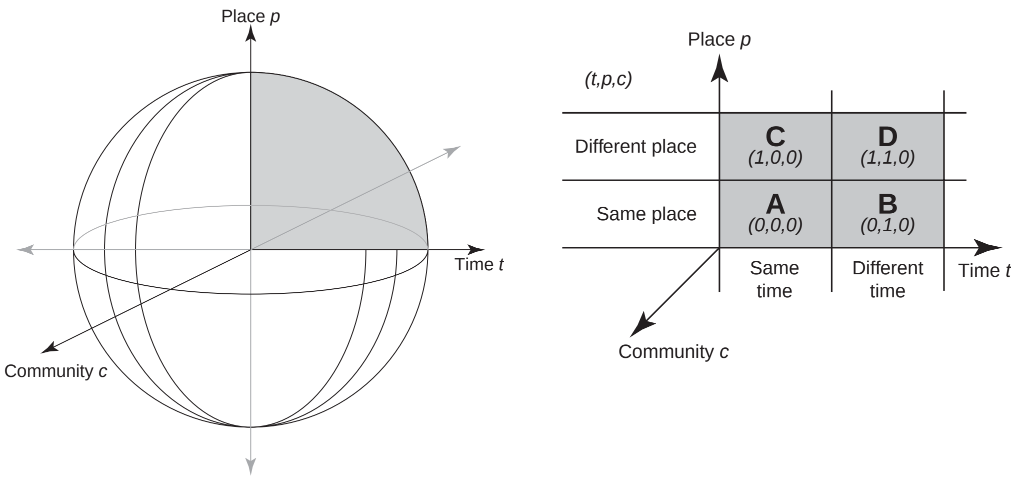


Figure 2.1: Distribution over space, time, and community. In the matrix on the right, only distribution over place and time is shown in the same community. The matrix hence represents the grey layer $(p, t, 0)$ in the 3-dimensional distribution space shown on the left.

B) in the matrix on the right in Figure 2.1 [Braun et al., 2001a]. We then refine iBistro's features to support distributed meetings (Cells C and D). Due to our focus on **global** software development teams, asynchronous communication is a central issue in iBistro (Cells B and D).

The distribution scenarios shown in the distribution matrix in Figure 2.1 (A, B, C, and D) all take place in the same community. As we are specifically interested in supporting international projects, iBistro has to support different communities as well. By different communities in this context, we understand local and remote groups who, while building a project team by working towards the same project goal, do not share cultural and social similarities and properties between them. Community distribution would take place in Cell $C' = (1, 0, 1)$ and Cell $D' = (1, 1, 1)$, where vector $(p, t, 0)$ denotes for instance the local community, while vector $(p', t', 1)$ denotes the remote community. (Omitted in the matrix shown on the right in Figure 2.1.)

2.2.2 LEVELS OF COOPERATION

Different distribution scenarios require electronic support for teams to cooperate. Teufel [Teufel et al., 1995] classifies the level of support in electronic tools by considering the intensity of cooperation. The intensity of cooperation is classified into *Communication*, *Coordination*, and *Collaboration*. (Called the *"3C"-Model.*)

The lowest level of support is provided by *communication* tools, such as email, or video- or teleconferencing. Communication can take place synchronously, such as in a chat, meeting, or telephone conversation, or asynchronously, such as in regular mail, email, or voicemail. Communication can take place with or without physical presence. Communication represents the lowest level of support, as it requires only an infrastructure for content exchange, regardless of what is actually transmitted. Beyond this, various *communication vehicles* or *communication media*, the means of communication, provide point-to-point communication, multicast communication, and broadcast communication.

Systems that help to coordinate the work of teams or individuals, such as project management tools, are called *coordination tools*. While there are some project management and coordination support tools, such as schedule planning tools in Microsoft Outlook or Lotus Notes, it is worth mentioning that many project management tools that are actually used are not necessarily electronically based (such as the project managers's calendar and notes book etc.). They are to be considered in the design for a (informal) meeting support tool.

The highest level of support is provided by *collaborative* tools, such as group editors or electronic meeting rooms. Collaborative tools support several teammembers to work together on the same artifact of work. A group editor, for example, allows several authors to work on the same document while tracking the changes made by each individual author and managing concurrent changes on the same portions of the document.

Figure 2.2 shows the 3C-Model.

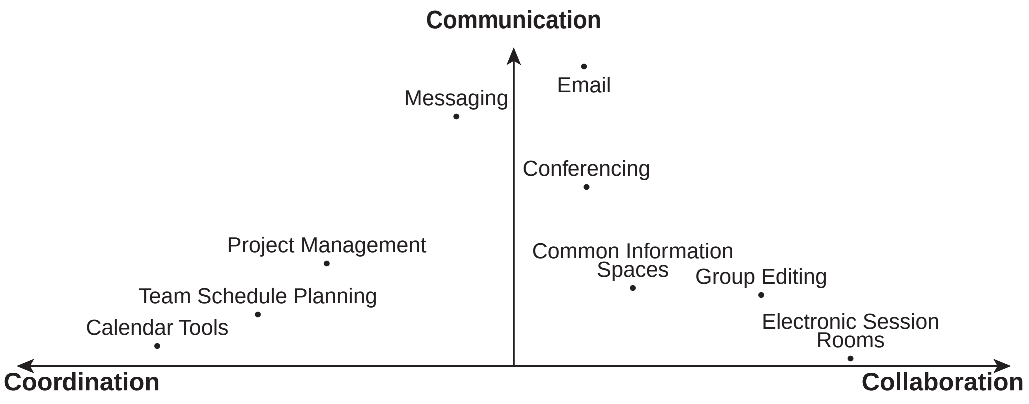


Figure 2.2: Classification of CSCW systems according to the *3C-Model* and related tool support (taken from [Borghoff and Schlichter, 2000]).

2.3 DIMENSIONS OF DISTRIBUTION

In distributed development we distinguish *three dimensions of distribution*: Geographical distribution, temporal distribution, and communication crossing community borders (community distribution).

Geographical distribution occurs in projects where the distance between sites as well as the amount of different sites negatively impacts on an organization's communication and information exchange. Two reasons for this are that distance complicates sharing resources and (technical) infrastructure and with increasing distance the possibility of meeting in person is reduced.

Temporal distribution affects teams working in different time-zones, such as on different continents or different work shifts. Temporal distribution is a result of distance or shift-work. If the working hours of sites overlap, synchronous communication is possible, for instance in face-to-face meetings or telephone calls. With none or little overlapping hours, asynchronous media, such as email or fax, has to be used, resulting in much higher latency.

Community distribution affects people in a community who do not necessarily know each other or interact on a personal basis. In the case of global companies and world-wide staffing, people from different organizational cultures work together in a single project. Each group may have its own language, terminology, tools, and methods, making collaboration difficult.

Geographical distribution, depending on the distance, also may entail temporal and community distribution, for instance if an organization is distributed over two continents. Human project communication is party affected by the company culture, team composition, community composition and other aspects that result partly from the distribution scenarios.

2.4 GLOBAL SOFTWARE DEVELOPMENT TEAMS

Global software development teams access complementary resources from multiple countries. They provide a structured partnerships in different countries and leverage capabilities of each center to create cost-effective solutions. We introduce the following definition for a global software development team:

Definition. A *global software development team* (GSD team) comprises the involvement of at least two collaborating teams with geographical distribution so that (1) different social groups of people working in the teams are involved, (2) the resulting time-shift allows no or little synchronous work (temporal distribution), and (3) communication is supported by electronic means.

The main issues for CSCW in larger organizations are related to communication. Borghoff and Schlichter identify three communication issues in larger organizations [Borghoff and Schlichter, 2000]:

1. inefficient internal communication
2. restricted internal and external communication possibilities
3. limited information technology

In the following, we will introduce a software architecture to address the issues for GSD team support.

2.5 THE STANDARD BLACKBOARD MODEL

Blackboard style [Carver and Lesser, 1992; Buschmann et al., 1996; Bass et al., 1998] refers to a particular way of organizing computing systems in general. The name 'blackboard' was chosen because it is reminiscent of the type of brainstorming situations in which human experts sit in front of a real blackboard and work together to solve a problem. The problem can be split into several sub-problems, each belonging to an individual domain of expertise. The experts work together by applying the following rules: Every contribution to the problem made by the experts has to be written to the blackboard; the blackboard can only be used by one expert at a time (no concurrency); the experts are only allowed to communicate with each other via the blackboard. The experts may use additional resources, for instance a (private) library, to work on their sub-problem.

The primary domain for blackboard systems is to solve problems where no deterministic solution strategies are known. In such cases, only patchy knowledge about how to tackle the 'overall' problem is available and expertise from various domains is needed. In the blackboard system, several specialized sub-systems, also called *knowledge sources*, assemble their knowledge. Each knowledge source contributes to the problem solving process by adding its specific expertise to the blackboard similar to the process of several human experts sitting around a table and adding their individual expertise. The knowledge sources assemble their knowledge to build a possible *partial* or *approximate solution*. The blackboard arrives at *layers of solutions*. The layers result from the level of abstraction of the individual contributions. A *layered blackboard* stores several levels of defaults (or *preconditions*), partial solutions and intermediate data, called *hypothesis*, and finally full solutions. Every hypothesis is evaluated for being reused to built a solution, or dropped.

The blackboard system is a *data-centered architecture*. Data-centered architectures [Bass et al., 1998] offer a solution to the problem of integrating data into larger systems in a structured manner. They are used in an environment were the access and update of widely accessed data is a primary focus. The structured approach enables adding, removing, modifying, and searching the data-centered architecture. By adding or removing components, data-centered architectures offer a scalable architectural style which is increasingly important for today's systems.

There are two different *coordination models* for data-centered architectures: the *repository* and the *blackboard* architecture.

A repository consists of an amount of independent clients built around a central shared data repository. In the *passive repository*, all activity is triggered by the (remote) components. The repository remains passive and waits for transactions which are initiated by components.

An *active repository*, such as a blackboard, can send notifications to components. Components can subscribe to data items or activities of interest and will be notified if a change occurs. The active repository initiates activity by calling the components.

The HEARSAY-II [Erman et al., 1980] speech understanding system developed at Carnegie Mellon in the early 1970s was the first system that was completely developed accordingly to the blackboard style. Another system, the HASP [Nii et al., 1982; Nii, 1986b] project at CMU, developed from 1973 to 1976, was built to interpret continuous passive sonar data.

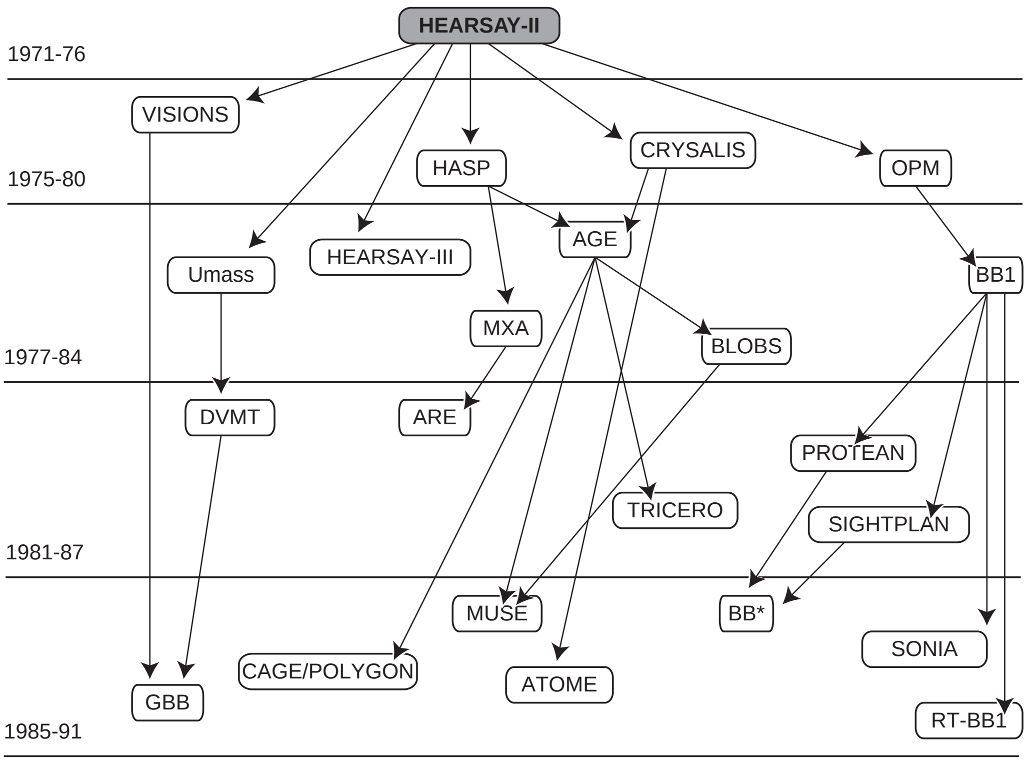


Figure 2.3: The Evolution of Blackboard Systems (based on [Drogoul, 2001]). All blackboard systems are based on the early research at CMU. The systems shown differ especially in their application domain, while still implementing features of the standard blackboard model.

After 1976, many other blackboard systems were developed based on the standard blackboard principles that were investigated and used in HEARSAY-II and HASP at Carnegie Mellon in the 1972s. Figure 2.3 shows the history and evolution of blackboard systems based on the early research at Carnegie Mellon University. The blackboard systems shown in the figure, while sharing common principles or architecture, differ only in their application domain. The blackboard systems were built for one specific application domain from scratch. The usability of one blackboard system in a different application domain was not in the scope of development [Nii et al., 1982]. As a consequence, no canonical abstraction of the blackboard system design has ever existed. All systems shown in Figure 2.3 use similar technology and architecture for different specific application domains. The application domain specific knowledge and expertise is hardcoded and embedded in the system. Hence, the application domain can hardly be changed.

Typically, three levels of detail are considered: *blackboard application* refers to a blackboard system that was actually designed for a particular task. A less detailed view that still abstracts some detailed features and implementations is referred to as a *blackboard framework*. The framework provides the basis to be refined in a specific application context. The *blackboard model* refers to the most abstract level. The term *blackboard architecture* is synonymous with blackboard framework, with a stronger focus on the conceptional view of the system architecture [Buschmann et al., 1996].

Figure 2.4 shows the basic blackboard model consisting of four major components: the **blackboard**, several independent **knowledge sources**, and central **control** and **strategy** instances.

In a repository architecture, subsystems access and modify data in a single data structure – the repository. The control flow is imposed either by the subsystems or by the repository itself. If the control flow is based on the state of the central data, the resulting variant of the repository architecture is called Blackboard. A blackboard sends notification to subscribers when data of interest changes [Bass et al., 1998]. The subsystems (or clients) of a blackboard system are called Knowledge Sources.

The blackboard is very general in what kinds of data (Data Items) it might store. Typically, there are three types of data items: Default Data, partial solutions (Hypothesis), and Solutions. Most data items are created and stored during runtime, such as events. However, a blackboard can also store static information created at compile-time. The data items are incrementally modified by the knowledge sources and developed incrementally towards solutions as the system executes.

Figure 2.5 shows a standard blackboard architecture with additional layers. Information at different levels of abstraction or detail is stored in different layers (sometimes also called levels) in the blackboard system. The layers are arranged to correspond to a particular abstraction hierarchy, such as a part-of hierarchy.

A special characteristic of blackboard systems is the possibility of smooth integration of bottom-up and top-down analysis. In *bottom-up analysis*, a knowledge source works on a number of data items on lower level layers and creates a new entry at a higher level. In top-down analysis a new entry in one level results from examination of higher level layers. Each layer stores a particular abstrac-

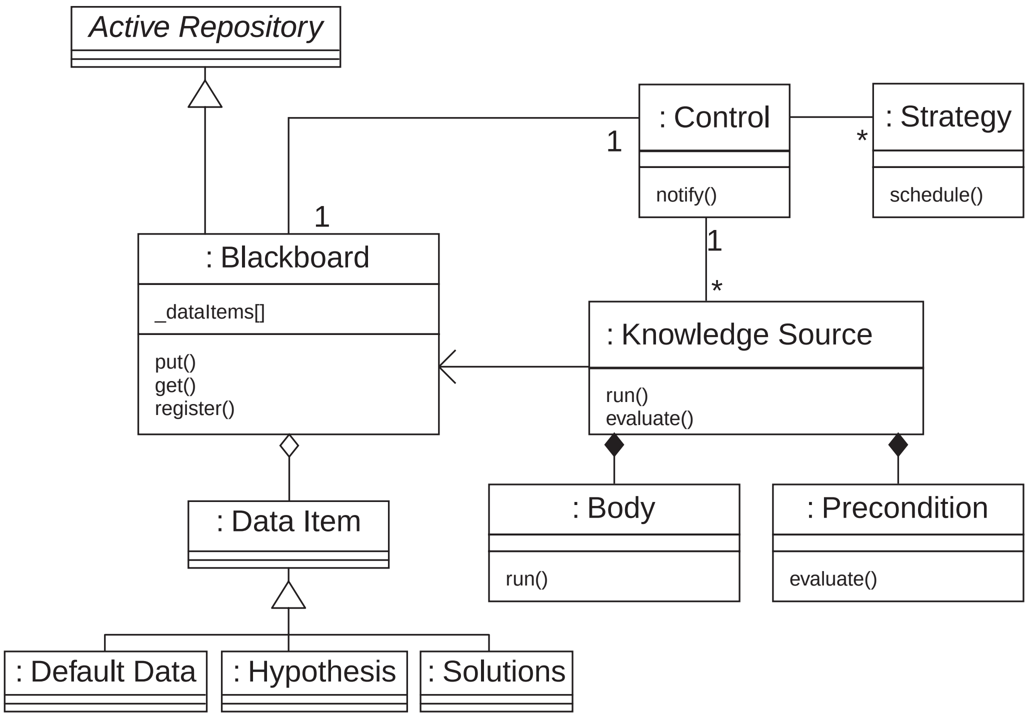


Figure 2.4: The standard blackboard model consists of the Blackboard which stores the Data Items, several Knowledge Sources, and a Control component that schedules the Knowledge Sources.

tion of objects, called Node. Layers represent classes of nodes. A node can have a number of attributes, such as a confidence factor or time-out value to limit computational time. A node can also be linked to other layers.

If a blackboard system requires more flexible partitioning of data, multiple *blackboard panels* (not shown in Figures 2.4 and 2.5), each containing its own layers, are used. This organization is needed if a blackboard system needs more than one hierarchy, for example, if a blackboard system combines two or more independent areas of expertise.

Two concepts are used to record the history of knowledge source scheduling in a blackboard system. *Event lists* are used to log all or specific classes of data item manipulation. In some systems all changes made to the blackboard are treated as events and are recorded [Nii, 1986a]. Such events are also used to schedule knowledge sources. The events are classified into categories of discrete sets and used as part of the control strategy. The control strategy may use the history of events in the blackboard to balance the execution of knowledge sources.

History lists record the history of data manipulation on the blackboard. In contrast to recording *what* changed, history lists log specific information on how

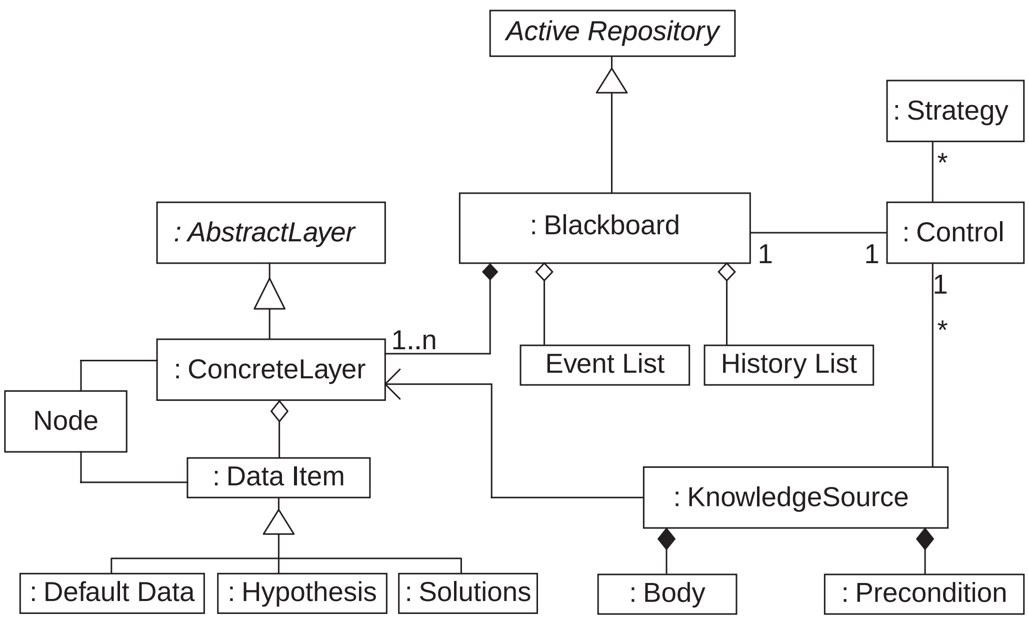


Figure 2.5: In the layered blackboard architecture, the blackboard is organized in several layers of abstraction. The knowledge sources consist of a precondition which is executed by the control. If the result is 'true', the knowledge source's body is executed on a specific set of data. (Attributes and operations are omitted.)

that change came about, such as knowledge sources involved and any variable and surrounding information and context that influenced the flow of events. History lists are typically used for debugging and system evaluation. History lists store the complete problem solving steps in a specific blackboard system and are therefore usable for development of new or improved systems and knowledge sources.

Most aspects of the blackboard architecture, such as organization of layers and panels, are static. Knowledge sources are the flexible part in the blackboard model, as they can be easily exchanged, added, or removed to use the system in a different context or improve the system.

The knowledge sources are computational entities which embody domain specific knowledge. Knowledge sources work exclusively with data items stored in the blackboard. The following properties characterize the most important rules for knowledge sources in the blackboard model:

- Knowledge sources register to the system and wait for execution. The knowledge sources and the reasoning style of the system itself is no systematic process and not predictable. Hence, it is an *opportunistic* process.

  An opportunity is a favorable circumstance that arises at just the right moment. Its occurrence is not predictable. In problem-solving situations, an opportunity presents an unanticipated chance to make progress towards a goal. To take advantage of opportunities as quickly as they develop, a problem-solver must have the ability to change course rapidly during its exploration of the developing search space. Sufficiently flexible systems pay close attention to the world and to the effects of their own decisions to facilitate quick reaction to both expected and unexpected developments. These systems are generally described as exhibiting opportunism in their control strategies, by continually redirecting their attention to the most urgent or promising issues at hand [Erman et al., 1980; Smith et al., 1990; Carver and Lesser, 1992]. Less flexible systems tend to wait until a previously determined and possibly extensive course of action has completed before they pause to update their current focus of attention.

- Due to the opportunistic order of knowledge source execution, the computation in a blackboard system does not follow any predictable or predefined order. Hence, the order of knowledge source execution and therefore the development of data items stored in the blackboard might develop in different paths for each (nondeterministic) execution time. The order of execution and type of reasoning is determined at run-time based on the current state of the blackboard, by previous knowledge source invocation, and by external events.

- Knowledge sources are not allowed to communicate directly with each other. All communication runs through the blackboard. This also implies that knowledge sources have no awareness of other knowledge sources connected to the same blackboard. However, a knowledge source can be part of two or more blackboards.

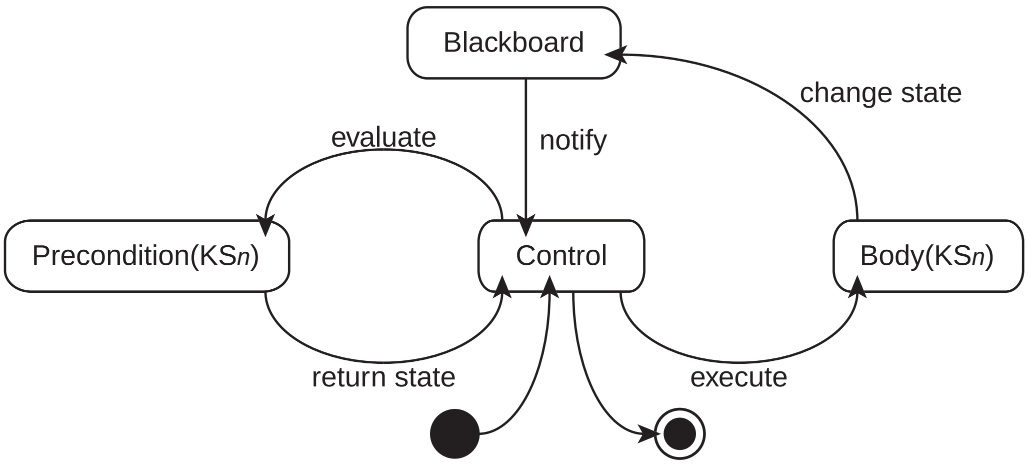


Figure 2.6: The dynamic model of the standard blackboard architecture.

Figure 2.6 shows the dynamic behavior in the standard blackboard model: The *Control* class schedules the knowledge sources. The Control class is notified on Data Item modifications by the Blackboard. Knowledge sources have to register with the Control class. The Control then iterates through the list of Knowledge Sources and evaluates their *Precondition* with provided data item as input. The Precondition can be seen as a rule which returns *true* if the given data item meets the precondition's expectation, and *false* otherwise. If the Precondition returns true, the *Body* (which is the code) of the knowledge source is executed.

While the Blackboard itself only knows about changes of Data Items to notify the Control, the Control and Knowledge source components have a restricted view on the overall process of knowledge acquisition. Knowledge sources work on one or two levels of abstraction (= levels or layers in the Blackboard). To direct the overall process of knowledge building and finding solutions, the Strategy component is used by the Control to provide a global view ("big picture") of the process seen over all levels of abstraction.

2.5.1 THE BLACKBOARD PATTERN

The blackboard model is an architectural pattern to design larger software systems. Complex software systems, especially business software, cannot be structured according to a single architectural pattern. Several system requirements have to be addressed by specific and suitable (architectural) patterns. The resulting combination of several patterns builds up a structural framework for a software system that needs further specification and detailed implementation.

In pattern-oriented software development [Buschmann et al., 1996], the blackboard is specifically listed as an architectural pattern. The architectural pattern represents the highest level of abstraction in the system of patterns. In a top-down approach, architectural patterns are used to specify the (initial) fundamental structure of the system. The blackboard pattern, for instance, belongs to a group of three architectural styles[1] that enable what Buschmann calls the transition from "mud to structure".

The rationale for pattern-oriented development lies in the perception that approaches from the application domain will not work sufficiently for technical aspects of the system for two reasons in particular: First, because a big software system consists of many components that do not have a direct relationship to the application domain (examples are for instance manager and helper functionalities). Second, we expect more than just a system that meets functional requirements from the application domain only. Many (non-functional) requirements and qualities have to be met, such as portability, maintainability, understandability, and stability [Buschmann et al., 1996] that are not directly related to the

[1]The other two styles are the layers pattern and the pipes and filters pattern.

application domain but to the solution domain.

2.5.2 THE BROKER ARCHITECTURE FOR BLACKBOARD SYSTEMS

A *broker architecture* [Lyons, 1991; Borghoff and Schlichter, 2000] serves as the basic communication model in the standard blackboard model. The generic broker architecture consists of several servers which register their exported services at a *broker*. Clients can call the broker for procurement of a suitable service. In principle, two categories of brokers are distinguished: *white pages* are brokers which deliver a suitable server by its name. This is used in nameservers, for instance. *Yellow pages*, in contrast, suggest a suitable server based on the type of service offered.

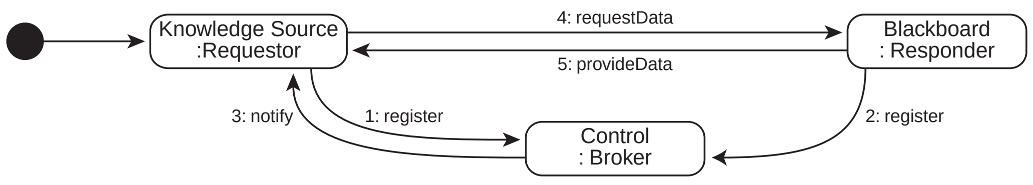


Figure 2.7: The dynamic model in a generic broker architecture (collaboration diagram).

Figure 2.7 depicts dynamic model of the broker architecture used in the blackboard model. Two components (`Requestor` and `Responder`) register themselves at the *Broker* by providing information on what type of service they deliver (export), and what type of information they are interested in (Steps 1 and 2 in Figure 2.7). The broker notifies the requestor upon availability of the requested type of service or information (Step 3) and where this information is available. The requestor then directly contacts the provider of the information (responder) and asks for the information (Step 4), which is then delivered by the service (Step 5).

2.6 EMPIRICAL METHODS IN SOCIAL RESEARCH

The communication study (see Chapter 4) was planned and carried out as a retrospective study to analyze the actual team work in a large industry project. We investigated how project teams communicate and cooperate, how they assess the communication and cooperation, and what communication vehicles they used. To study the non-technical and human aspects of team interaction in software development projects, new research methods are needed in a challenging area of research. Methods from social sciences have been developed and applied in many other disciplines. Qualitative methods are used to handle the complexity and richness of issues involving human behavior. We start by presenting some qualitative methods for data collection and analysis and describe how those methods can be used in empirical studies of software development in general, and how they are used in this dissertation in particular.

2.6.1 QUALITATIVE VS. QUANTITATIVE METHODS & DATA

Qualitative methods are based on the interpretation of human language and behavior. Qualitative analysis offers a complete and detailed description of all phenomena. In contrast to quantitative methods, they include even rare phenomena which receive the same amount of attention as frequent ones. Ambiguities which are inherent to human language [Seaman, 1999] are recognized in the analysis (text analysis). However, qualitative studies can not be extended to a wider population (for instance, extension from software development to generic development) with the same degree of certainty the quantitative analysis can. Further, the findings can not be tested for statistical significance easily.

Quantitative analysis produces statistically reliable and generalizable results by classifying and counting features of a given set. The findings can be generalized to a larger population and can be compared directly. Quantitative analysis relies on valid sampling and significance techniques, thus, it allows to discover which phenomena are likely to be genuine reflections of the behavior of a team or variety, and which are merely chance occurrences [Fuchs et al., 1978].

In other words, the difference between qualitative and quantitative data is how the information is elicited and represented. Quantitative data is elicited and represented in numbers or other discrete categories, while qualitative data is recorded in text or pictures. A typical misunderstanding of qualitative and quantitative methods is that qualitative data is considered to be subjective, while quantitative data is thought to be objective. Essentially, the type of data, either quantitative or qualitative, only affects the way the information is stored (qualitative data as text versus quantitative data as numbers), not its (relative) subjectivity or objectivity.

2.6.2 QUALITATIVE METHODS FOR DATA COLLECTION

Data collection methods, as well as studies, can be divided into prospective and retrospective. *Prospective* investigations are started early, for instance at project start, and accompany the project. In a prospective study, future measures are part of the investigation and are reflected in the study. In a *retrospective* study, the project is reviewed after its actual end, or after reaching a critical milestone. No measures during the survey are adapted to the investigation dynamically.

Participant observations have to be planned and set up in advance and are transacted during an ongoing project. Interviewing techniques, in contrast, can be both executed during an project or in hindsight. For capturing as much insights and experiences, interviews are executed retrospectively after the project or an important stage ended. For capturing a maximum of knowledge, prospective and retrospective approaches can be combined.

Participant Observations

Participant observations (sometimes also referred to as ethnographic observations), refer to "research that involves social interaction between the researcher and informants in the milieu of the latter, during which data are systematically and unobtrusively collected" [Seaman, 1999]. The results from observing participants (or developers) in a software project is limited, because much of software development work takes place inside the heads of developers. Additionally, large software projects often are lengthy with many involved people. This makes observations practically difficult. However, as on today's projects much work is done in a PC, new ways of technical observation could be used (we introduce communication and software metrics later on.)

In the following, we introduce observation techniques that are used to collect information during software development (see [Taylor and Bogdan, 1984; Seaman, 1999]):

Think aloud protocols are often used in usability testing. Users have to verbalize the thought process so that observers can understand the process going on. There are a number of issues with observational techniques. First of all, observers should be as unobtrusive as possible ("like a fly on the wall"), in order not to affect or bias the observed phenomenon. Our appraisal of observational techniques, hence, is that observers definitely do affect the observed phenomenon, though positively in a sense that participants try to give a good impression of their professionalism and behavior.

Field notes are used to write down observational records. Observers start to fill in field notes as the observation starts. Field notes are later augmented with as much of the information that can be remembered. Any information relevant for the observed phenomenon should be included, for instance place, time, participants, topic. Thus, the notes are very time-consuming to record and evaluate. Field notes also contain subjective ratings by the observer, such as an individual assessment of the project or company situation. Hence, field notes have to be kept confidential throughout the whole investigation.

In *rater agreement exercises*, a second observer randomly attends meetings to compare the field notes of the two observers and make sure the same amount of information is captured in all meetings and by all observers. Moreover, *codes* are used to categorize discussions or meetings. Briefing by the principal observer helps the observers to deal with lack of familiarity with the project and lack of experience with the instrument (i.e., coding categories). During the rater agreement exercise, the investigative setup is checked for accuracy and that data collection techniques used are confirmed being robust. Rater agreement exercises are used to evaluate the validity and consistency of data collected during participant observations. Further, rater agreement exercises are used to ensure that an observer records data in a way that is not only understandable to him or her and that this information is normalized against bias.

Electronic capture of observed meetings and interviews is based on combined audio and video capturing. Electronic recording ensures that all information of a meeting is recorded by recording the whole meeting. If electronic meeting capture is used, the field notes can be written after the meeting takes place. In this approach, the observer can take as much time as is needed to write down his or her observations. Any passage that is unclear in the first cycle can be repeated. Inaccuracies are much less likely to be introduced. It is worth mentioning briefly, that audio/ video capture might experience resistance from some users. In that case, planning for alternatives that still enable comparability of field notes is necessary. (This principle is also used in iBistro for post-mortem meeting annotation. See Chapter 5.)

Interviewing

Interviewing is a method for data collection, where a trained investigator (interviewer) directly asks oral questions to a subject (interviewee). Interviewing is commonly used for retrospective collection of qualitative data. In an interview, historical information (for instance at the end of a project) is elicitated from the interviewee's memory. Interviews are specifically suitable to collect opinions or impressions of the observer, while observational techniques reflect more the observer's view of a meeting. However, interviews are sometimes combined with observational techniques to clarify or elicit the impression of a meeting. Interviews are also used to identify special knowledge or terminology used in a particular setting.

1. *Unstructured interviews.* In an unstructured interview, the interviewee is both the source of questions and answers. The objective is to elicit as much

qualitative information as possible by asking open-ended questions. Thus, the answers are hard to classify.

2. *Structured interview.* During a structured interview, the interviewer has very specific objectives which are addressed by asking a sequence of prepared questions. The sequence is stipulated in an interview guide or interview form[2], which has to be used for every individual interviewee. Structured interviews are also used to collect quantitative data.

   Figure 2.8 is an example of a structured interview with an interviewer following directions given in the interview guide.

Introduction:		"In the following stage, we will present you with a list of communication vehicles. At the end, you will have the possibility to comment on this vehicles or name additional means of communication used. For every means of communication, we are particularly interested in the following three questions: (a) How often did you use this communication vehicle? (b) Would you rate the amount of usage of this vehicle as used too often, just right, or too seldom? (c) How do you rate this vehicle accordingly to its effectiveness in solving problems within your context? Do you have any questions at this point?"
1.	**Question:**	"How often did you use email?"
	Answer:	"Well, I used it quite often."
2.	**Question:**	"How would you assess the amount of email usage?"
	Answer:	"Just right"
3.	**Question:**	"How do you rate the effectiveness of email communication in solving problems within your context?"
	Answer:	"Very effective"

Figure 2.8: Example for a structured interview, conducted by an interviewer following specific questions listed in an interview guide. The answers given by the subject ("Answer:") are captured in field notes.

The study and field notes must be designed flexible enough to record unforeseen information, such as other means of communication used by some interviewees, or additional information on specific questions. For instance, some interviewees in the study introduced in Chapter 4 used the Windows `net send` shell command and instant messaging tools to send short screen messages to a colleague's computer. Some developers reported, that especially during assembly testing this turned out very useful. While this infor-

[2]The interview guide for the communication survey introduced in Chapter 4 is shown in the appendix.

mation was expected in the design of the structured interview, this valuable information is to be recorded.

3. *Semi-structured interviews* include a mix of structured and unstructured interviews, i.e., a combination of open-ended and specific questions. This approach is chosen when much relevant but unforeseen information is expected.

   Figure 2.9 is an example of a semi-structured interview with an interviewer following directions given in the interview guide and following up on unexpected answers.

1.	**Question:**	"How often did you use email: very often, often, seldom, or never?"
	Answer:	"Actually, we used network chat utilities instead of email in the team."
2.	**Question:**	"What chat utilities? This is interesting. Why did you prefer those tools over email?"
	Answer:	"We used Messenger, because it is more interactive and easily includes more people in the communication loop – compared to email."

Figure 2.9: Example for a semi-structured interview. The interviewer starts with specific questions listed in an interview guide. When unexpected or interesting answers are given by the subject, the interviewer may follow up on them.

4. *Standardized interviews* are a specific form of an interview where all details of the question and response behavior of the investigator is predefined to achieve a maximum of comparability. This form of interview is especially well-suited to verify a hypothesis.

Figure 2.10 is an example of a standardized interview.

1.	**Question:**	"How often did you use email: hourly, several times a day, daily, every second day, or weekly?"
	Answer:	"several times a day." *(The interviewee is only allowed to select an answer from the predefined set of answers, shown as underlined text.)*

Figure 2.10: Example for a standardized interview with predefined answers.

Interview planning. To achieve comparable and reliable results, planning an interview is an important step. During interview planning, the goal for the investigation and approach is defined. Depending on the type of interview and level of formalism, field notes or an interview guide have to be prepared. Field notes are a more formal way of recording the information, typically by filling in forms. An interview guide, while being less formal, helps the interviewer organizing the interview. It contains a list of all open-ended questions, with notes about directions and maybe arranged with some “if-then” conditions, which lead the investigator along several paths depending on the answers to previous questions.

Interviewer briefing. As some of the interviewers are often not familiar with the interviewing approach, the project under investigation, or other details which an interviewee might confront them with, interviewer briefing permits the interviewer to understand subjects. During the briefing, the goal for the investigation is clarified, the interview is planned, the forms are explained and introduced, measures are taken to ensure comparability and equality of interviews, interviewing style is checked and trained, and dry-runs are performed to ensure comparability both between several interviewers and between the first interviews held versus subsequent interviews of a single interviewer.

A full description of interviewing techniques can be found in [Fuchs et al., 1978; Friedrichs, 1984; Guba and Lincoln, 1985]. Interviewing with special focus on empirical software engineering is found in [Seaman, 1999].

2.6.3 COMBINATION OF QUALITATIVE AND QUANTITATIVE METHODS

If a combination of qualitative and quantitative data is used in a study, the related qualitative data has to be transformed to extract values for quantitative or statistical analysis. This process is called “coding”.

During coding, qualitative data is classified into quantitative variables. Qualitative data is information typically expressed in text, while quantitative data is represented as numbers or other discrete categories.
Figure 2.11 shows an example for coding:

2.6.4 DATA ANALYSIS METHODS

There are two possibilities for data analysis:

1. Generate a hypothesis that fits the data (or is “grounded” in that data).
2. Build-up the “weight of evidence” necessary to confirm the hypothesis.

In the first case, a hypothesis is generated *after* the survey. The results of data analysis are used to phrase a hypothesis that best describes the elicited information. This hypothesis is then refined and verified using the second possibility (2) for data analysis. The results of the survey are used to assess and evaluate a given hypothesis. The hypothesis might arise from theoretical reflections, practical experience, or from an earlier study.

Introduction:	"Let us have a look on the proportion of communication with people *outside* your own team boundaries and communication with people *within* your own team."
Question:	"Did you communicate only internally, mostly internally, half-half, mostly externally, or only externally?"
Answer:	"Well, I usually try to fix problems with the responsible expert. However, of course I ask my team mates first, as they are located in the same room. This typically answers most questions. . . "
Interviewer:	So what would you say was your communication pattern accordingly to the categories mentioned?
Answer:	Mostly internal.

Figure 2.11: Example for coding during a semi-structured qualitative interview. The spoken (and potentially captured) textual explanation has to be classified into quantitative values (i.e., "only internally", "mostly internally", "half-half", "mostly externally", and "only externally"). This is achieved through interpretation, which has to be protected against potential bias.

2.7 QUANTITATIVE METRICS

Quantitative metrics focus on samples that are directly measurable with metrics. In this dissertation, three categories of quantitative metrics are taken into consideration: software metrics, communication metrics, and project and performance metrics.

Software metrics [Perlis et al., 1981; Grady and Caswell, 1987] are measurable directly from the artifacts of software development, typically from source code. The goals for software metrics are to assess the complexity or programming 'quality' of a given source code. If certain measured values exceed a defined range of values, they are used as indicators for bad programming style (such as "spaghetti code"), or to identify exceedingly complex parts of the system.

The measurement of software recognizes an increasing importance with the generic growth of software systems. The measurement of functionally decomposed software has been studied with great success for a while. Today, many software projects, including the project under investigation in Chapter 4, deal with object-oriented development. Metrics for object-oriented systems [Henderson-Sellers, 1996] are more challenging and error-prone for several reasons: The emergence of object-oriented programming languages was accompanied by new programming paradigms and new levels of abstraction. Especially the measurement of complexity of object-oriented programs is more difficult than that of functional/ procedural programming languages. In particular, the use of 'good' object-oriented programming techniques, such as the usage of design and architectural patterns, are challenges for measurement of object-oriented programs.

Communication metrics result from the measurement of communication artifacts, such as in groupware tools, email, etc. Metrics on communication artifacts can be used to gain significant insight into the development process that produced them [Dutoit and Bruegge, 1998]. The advantage of communication metrics is that they are independent of the implementation technology and development infrastructure used. Communication metrics can be used to construct and visualize communication paths, including timestamps, stakeholders, communicating peers, and other interesting traces in project communication. Communication metrics can also be used in distributed projects.

Communication, in particular informal communication, is difficult to measure. This is crucial, as personal and informal communication is considered relevant for project success in particular [Kraut and Streeter, 1995]. Moreover, in many organizations and companies, communication metrics cannot be captured due to limitations imposed by the works council and specific laws, especially the Works Constitutions Act and Data Protection Act[3]. This applies to most, if not all, countries within the European Union.

While many of the technical and application-domain specific properties can be defined from a technical point of view, an approach to finally implement the system in an organization has to regard for instance user acceptance and usability.

Some of the attitudes towards change in general are well described in social sciences. Two definitions which are consecutively used in the communication survey (Chapter 4) and in evaluation and lessons learned (Chapters 7 and 8) are value orientation and action orientation.

Value orientation is related to aspects of the orientation of an actor which oblige him or her to consider certain norms, standards, options, and criteria anytime the actor is in a situation to choose. The value orientation within a certain culture (for instance, society or community) tends to build-up a structured system. [Friedrichs, 1984]

In other words, inherent and implicit values are known to influence people's attitude and decisions, whereas the individuals are often not (fully) aware of the influence of those values.

The term *action orientation* denotes considerations for the behavior of an actor towards a social, physical, or cultural object. [Friedrichs, 1984]

Table 2.2 shows an overview of methods used in empirical software engineering that are used in this dissertation.

[3]see *Directive 95/46/EC of the European Parliament, Official Journal of the European Communities of 23rd. November 1995 No L. 281 p. 31.*

2.7.1 OVERVIEW OF EMPIRICAL METHODS

Participant Observations	
Think aloud protocols	Interviewees have to verbalize the thought process of what they are doing. An unobtrusive observer watches the interviewee and makes notes. Often used in usability testing.
Field notes	Field notes are used in observational methods to write down relevant observations. Field notes are written during an observation and augmented later. Field notes will always contain subjective ratings of the observer, hence they are to be kept confidentially.
Rater agreement	A *rater agreement exercise* is used to evaluate the validity and consistency of data collected and to make sure that all records (field notes) are understandable to people other than the original observer. They are, more generally, used to categorize interviewees, identify possible bias, and classify field notes/ interviewees after they were recorded.
Electronic observations	Electronic observations ensure capture of a maximum of information. Field notes are typically extracted later, based on the electronic (video and/ or audio) capture.
Interviewing	
Unstructured	In an unstructured interview, the interviewer asks open ended questions (without an interview guide) to elicit as much qualitative information as possible.
Structured	In a structured interview, very specific objectives are addressed by asking prepared questions. An interview guide, listing those questions, is used.
Semi-structured	A mix of structured and unstructured interviews, for instance a combination of open-ended and specific questions. Use when much relevant but unforeseen information is expected.
Standardized	Any interview type (structured, unstructured, and semi-structured) can be standardized to achieve a maximum of comparability and reduce bias. In a standardized interview, all detail of the interviewer's behavior is predefined. Standardized interviews are in particular used to verify a hypothesis.
Combination of Qualitative and Quantitative Methods	
Coding	The process of classifying qualitative data into quantitative sets is called coding. The challenge is to identify the quantitative variable that fits best the information described by qualitative text.
Data Analysis	
Generation of theory	Draft a hypothesis out of the elicited data. The hypothesis is created *retrospectively*.
Confirmation of theory	Evaluate a pre-existing hypothesis. The study is executed *prospectively*.

Table 2.2: Overview of methods used in empirical software engineering.

CHAPTER 3

PROBLEM DEFINITION

A group of production workers at a NASA contractor had found a simple way to improve the calibration of the rocket engines. The foreman wrote a memo with his suggestions to his superiors two years ago, but nothing happened. When he asked why, he was told his suggestion was too expensive. "Too expensive to paint four little lines?" *He said in disbelief. They all laughed,* "It's not the paint, it's the paperwork. They would have to revise all the manuals."

[RICHARD P. FEYNMAN]
in "Personal Observations on the Reliability of the Shuttle", 1988.

In the following, we consider software development in industrial and related organizations, and then progress to specific problems resulting from the various levels of distribution. Finally, we discuss formal and informal project communication and roles in software development.

3.1 DISTRIBUTED SOFTWARE DEVELOPMENT

Software development is no isolated activity of individual programmers, like it might have been in the past. While single programmers could initially handcraft small programs, the workload in industry projects soon grew beyond the amount that could be handled by individuals. Especially business software systems are sometimes developed by huge teams. The success of managing teams of a dozen or more professionals was mixed. Regardless, the scale of development work still grows. Watts Humphrey observed, associated with his work on the Capability Maturity Model (CMM) for development organizations that "the history of software development is one of increasing scale" (in [Booch, 2000]).

As companies and applications grow and become more complex, industrial teams have to deal with increasing size, complexity, cost, and time to market pressure. The source code of Windows 2000, for instance, has more than 30 million lines of code in the base operating system. Windows NT 4.0, in comparison, had 16.5 million lines of code [Johnson, 1998]. Software developers experience predefined requirements specifications, schedules, and release dates. On the other side, software development is increasingly complicated due to the sheer size of the projects and the resulting growth in headcount and length of projects. As a consequence, organizational models were adapted and social aspects in working together and soft issues become increasingly important.

Software companies typically address social aspects and resulting challenges with a flat hierarchy. For instance, at Accenture, a project manager reports to some kind of executive director. From the project manager downwards, positions in the hierarchy are sub-project managers, team leads, and team members. The teams are supported by human resource representative staff and team assistants. To bring teams up to speed, team building activities are dispatched to constitute a sense of "teamness" to disburden the synergy in teams.

When systems and application software become bigger and more complex, the modules are decomposed into smaller parts, called subsystems, to handle complexity. If the resulting subsystems are split to be processed by separate programmers or teams, subsystem decomposition leads to "division of labor". Individual developers work on the smaller parts which altogether build a larger component of the final system. This creates demand for distributed development. Some of the reasons for distributed development are for instance:

- Skilled developers and IT specialists are scarce and expensive in many parts of the industrial world. In countries of the third world and in emerging nations, skilled IT workforce is available at far lower wages.
- Software developers often prefer to work in a well-known social environment with little or no travel [Weinberg, 2001]. This is a seminal challenge, especially for consulting firms, which develop software as a general rule at the client's site, putting up with additional cost and workforces working out of town.
- Software companies increasingly work in a competitive market place with short product cycles, cost, and time to market pressure. Similar to other industries, for instance production and manufacturing, this leads to global development. In such a scenario, software development "following the sun" and therefore leveraging the different time zones around the world for the purpose of some form of shift work seems to promise improvement.

Carmel in his book on Global Software Teams summarizes a typical project manager's perception of distributed development as follows: "No one in their right mind would do this." [Carmel, 1999]

The divide and conquer technique breaks a larger problem into smaller parts to be solved independently. The final solution is composed of the integration of resulting sub-solutions. Divide and conquer prepares the way for *subsystem decomposition*, where a large system is decomposed into smaller parts that can be handled by individual teams of developers. This is also a means of managing the development of software systems that are too large to be completed in time by one individual or few programmers by partitioning work and allowing for concurrent development. In this case, subsystem decomposition is accompanied by *division of labor* [Brooks, 1995], which made large systems' development a lot different from the development of smaller software. Large projects are split-up into many smaller sub-projects, components or parts. These sub-tasks then are assigned to loosely coupled development teams. While improving development speed by concurrent development and making large projects possible, division of labor introduces new organizational and managerial challenges project managers and software engineering researchers have to deal with.

Our claim is, that many of the challenges in distributed projects that are related to the fact of division of labor (e.g., organizing work in teams, communication, getting control for several concurrent teams, managing dependencies, ...) are similar to large projects. We claim that many advancements computer scientists and software engineers made to address division of labor also help with distributed development.

This dissertation presents the following hypothesis:

Hypothesis. Distributed development projects do not differ much from large software development projects due to division of labor.

In this thesis, we present strong anecdotal and empirical evidence that existing experiences from large software projects can be transformed into (globally) distributed projects, given a suitable team structure. We present an empirical survey done at a large software project to support this hypothesis in Chapter 4.

One example is the process of unit and assembly testing procedures versus system testing, which is used in large software projects to ensure that the individual parts of the system work as specified and interface correctly and that the final (assembled) system still works correctly as a whole (without side-effects) and satisfies the requirements.

Distributed software development is practiced in several forms. Three examples of distributed software development settings are detailed in the following sections:

First, we introduce **industrial approaches** to develop and run software remotely. Specifically, solution centers (near shore), offshore development, and outsourcing are discussed. Second, in the **open-source community** software is build by various developers around the world who communicate over the internet. Third, we discuss **virtual corporations** as an upcoming temporary organizational form for distributed software development.

Industrial approaches

The concept to develop software and build, test, and run solutions in dedicated *solution centers* is derived from development companies that build off-the-shelf software or reusable components. Development in solution centers can be considered distributed because clients and other stakeholders responsible for identifying and analyzing requirements work separated from the developers. Consultancies, in contrast, typically work and develop software at the client site. IT consultants avoid distribution and often even work in *mixed development teams* with both client and third-party personnel, as well as consultants working together in one team.

The approach provided by solution centers is promising for companies that produce client-specific software, too. Solution centers offer a "design, build, run" approach to fast, reliable, and recurring delivery of software or services. Solution centers also help to deal with specific organizational issues in software development that especially consultancies have to deal with. For instance, it is easier to staff a solution center than to hire personal that is willing to travel to client locations.

The focus of solution centers is on developing reusable solutions. Integration of legacy systems and using business critical, security relevant, or confidential material in solution centers is difficult. Further, projects that are pursued with intense client interaction, e.g., in projects where client personal works together with external personal in mixed development teams, can not be handled in solution centers. In such situations, only small parts that are loosely coupled with the development work can be outsourced to a solution center. In general, projects with changing or vague requirements and iterative life cycles are critical and hence still pursued at the client site.

Figure 3.1 shows the spectrum of distribution scenarios in professional real-world, single-site and distributed software development projects. Crossing one or more level of distribution – either spatial, temporal, or community – leads to distributed software development. Such distributed development scenarios also depend on the level of cooperation. This results in the following categories of distributed software development projects:

In *on-site development*, local members of staff develop software internally. This approach is the traditional approach in software development and still implement in many organizations. This approach is used for instance where off-the-shelf software is to be developed, or if the software developed comprehends specific unique selling propositions so that including external staff might be a

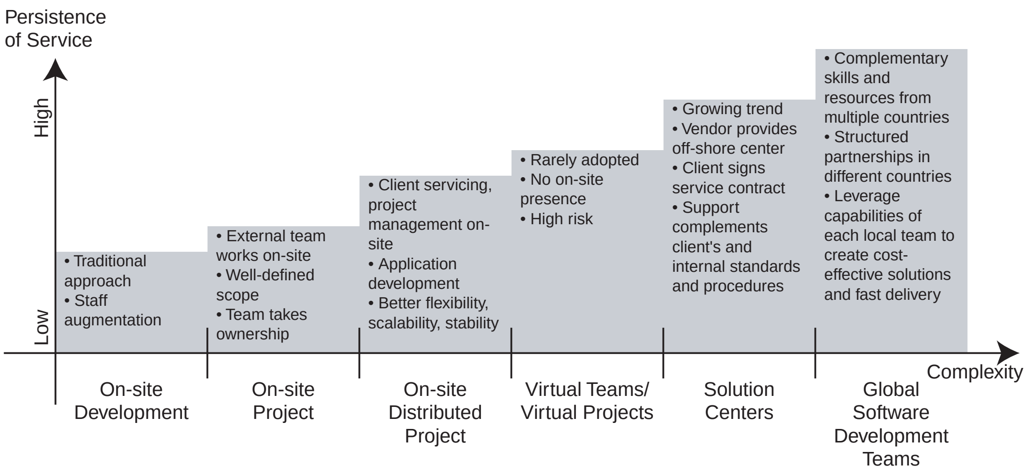

Persistence of Service
High
Low
Complexity
• Traditional approach
• Staff augmentation
• External team works on-site
• Well-defined scope
• Team takes ownership
• Client servicing, project management on-site
• Application development
• Better flexibility, scalability, stability
• Rarely adopted
• No on-site presence
• High risk
• Growing trend
• Vendor provides off-shore center
• Client signs service contract
• Support complements client's and internal standards and procedures
• Complementary skills and resources from multiple countries
• Structured partnerships in different countries
• Leverage capabilities of each local team to create cost-effective solutions and fast delivery
On-site Development
On-site Project
On-site Distributed Project
Virtual Teams/ Virtual Projects
Solution Centers
Global Software Development Teams

Figure 3.1: Levels of distribution in software development projects.

risk, for instance for intellectual property.

In an *on-site project*, external teams work on-site in a project-based setting with a well defined scope in which the team takes ownership. The local staff is augmented by adding external resources, such as freelancers or consultants, or a third-party contractor. The on-site project, however, is still single-site.

The *on-site distributed project* is a combination of local client servicing and on-site project management augmented with off-shore application development. The on-site distributed project offers better flexibility, scalability, and stability by adding external remote resources.

Virtual teams or *virtual projects* are rarely adopted in industry today due to high risk. Virtual projects are conducted without on-site presence.

Solution centers are a growing trend in software development. A vendor provides development capabilities in an off-shore center. A service contract between the vendor and the client defines the types and levels of service provided. The support provided by the solution center complements the client's and internal standards and procedures.

Global software development (GSD) teams are a vision introduced in this thesis and not a actual industrial practice. As defined in Section 2.4, a GSD team consists of at least two collaborating teams with geographical distribution where different social groups of people are involved. The global distribution also leads to time-shifts which allows no or little synchronous work. Communication is supported by electronic means.

The GSD team uses skills from multiple countries, for instance programmers in India and a requirements team at the client site in Germany. Each local team draws on specific resources. Structured partnerships, for instance with vendors and clients, are managed through the bigger organization of the GSD team to assist the local team. The goal of the GSD team is to leverage the specific capabilities of each local team to create cost-effective solutions and achieve fast delivery.

Offshore development denotes distribution scenarios where the software development work is performed at remote sites. In offshore development, the software requirements are still specified for one specific client (no "off-the-shelf" software). In offshore-projects, however, an on-site team works on the requirements specification and coordinates development, testing, and roll-out.

The software solutions developed *"offshore"* from the client site are still deployed to client-operated IT operations which install, run, and maintain the software. While the development stages of the software life cycle are outsourced, running the solution is still done in-house. *IT Outsourcing* closes the externalization loop started by solution centers, because an outsourced IT solution is also operated by an outsourcing partner in a third-party IT hosting. This includes all IT related work and service, including system operations and maintenance, responsibility for faults, upgrades and services. The properties of these services are defined in a *service level agreement*, which legaly defines guaranteed qualities of the service, such as response times.

Open-Source Software Development

Open-source software development naturally has to deal with global distribution of the development process. In open-source projects, distributed programmers develop parts of a larger program without ever meeting each other in person. The properties of a open-source project, however, differ a lot from commercial software development. Open-source projects typically implement technical software from the domain of computer science, for instance software development tools (compilers, editors), operating systems. Such programs are not implemented on user's requirements and many of the users are also (potential) users of the software (unlike in business software development).

Examples are GCC (the *GNU C Compiler*), a free compiler collection for C, C++, Fortran, Objective C and other languages, Linux, a UNIX-like operating system based on MINIX [Tanenbaum, 2001], or JBoss, a J2EE based application server written in Java. These applications written by the open-source community clone features of their commercial role models. Open-source development usually skips requirements elicitation. GIMP, for instance, an image-editing application, implements a subset of the professional image-editing standard software PhotoShop^{TM} by Adobe and skips its own requirements elicitation.

Testing in global distributed open-source projects is done explicitly by the developers, who are also users. The development process is incremental. New functionality is added upon user request without having a pre-defined schedule or list of functionality. Open-source developers and communities developed interesting and remarkable tools for communication and cooperation in global distributed software development. We discuss SOURCEFORGE^{TM} [SourceForge, 2001] as one example for cooperative platforms. SOURCEFORGE offers a variety of project-related services, such as:

SourceForge considers several project-related roles, such as an administrator, who is the initiator and kind-of project manager, developers (who can be sub-divided into several roles if needed), and users. SourceForge offers several services related to those roles. The Collaborative Development System (CDS) offers various web-based tools for the administrator role for maintenance of the project. CDS allows the administrator to post news items, administrate the developers (e.g., assign tasks or roles), post issues and ask for help, and manage the use of other services.

The single common point of contact for users and developers of a specific project is the Project Web Server. From this web page, further user and developer-related services can be accessed. Tracker, for instance, offers integrated support management tools for bug reporting, and support requests. Issues and requests can be classified and categorized by their importance. Mailing lists and discussion forums help developers stay in contact with their administrator and users.

SourceForge also provides services related to the software development life cycle, such as an Internet-based release service, shell services and a compile farm, test databases, version control (Project CVS Services), and a web-based service to increase the visibility of the project within the community (called "Trove Listing").

Virtual corporations

Virtual corporations are temporary networks of independent companies which are linked together by technical means to share skills, costs, and access to one another's markets. In a virtual corporation geographically distributed teams of each contributing companies are coordinated through electronic communications. The resulting organizational model is team-based, while the composition of the teams may cross the boundaries of the contributing companies.

The virtual corporation is a specific example of a *networked organization*. Smaller companies are realizing the benefits of being part of a virtual corporation, which can give them the benefits of the resources (e.g., financial resources, job opportunities, man power) of a large organization while retaining the agility and independence of a small one.

An example for a virtual corporation is when multiple companies working together for a single customer in a common project. Often, the single contractor or a leading company acts as the integrator, while different sub-contractors do the development. The sub-contractors add manpower and specified services to the project and are more or less exchangeable depending on the type of service they provide. These projects require a project based organization [Bruegge and Dutoit, 2003]. The assumption behind virtual companies are that the right mixture of skilled staff can be put together dynamically and relatively fast for a specific project, hence shortening product setup time, and lowering training needs. Under these conditions, a project can be staffed on- demand to help with employment related issues and balance headcount accordingly to the actual workload determined by the project.

3.1.1 PROJECT COMMUNICATION

Project communication can be classified by identifying the "level of formalism". *Formal* communication has a strong focus on the following *meeting attributes*: *agenda, process, schedule, location, participants, relationship*, and *objectives* [Braun et al., 2001a].

Depending on the type of meeting or communication, the meeting process might be predefined in detail, sometimes even by legal regulations (e.g., for a meeting of the supervisory board or for a stockholders' meeting the process, agenda, and audience is predefined). A formal meeting, for instance, typically has an agenda distributed in advance to all the invited participants. The meeting will have a specific schedule due to time constraints of the participants, and will take place at a predefined location. The participant's relationship results from social aspects, their company's culture, their hierarchical structure, their affiliations, etc.

A meeting is *informal* if the influence of one or more of the meeting attributes is reduced. This is illustrated by having a exemplary look on three different types of informal meetings:

Casual meetings in FreeWalk [Nakanishi et al., 1999] are held in a pleasant atmosphere with subtly hierarchical structures. Casual meetings are held without or with low ceremony (process), in casual attire, and often with beverages or snacks to support a convenient feeling. The expectations on the findings and outcome of the meeting are reduced.

Casual meetings are difficult to capture. They are often used as a kick-off to introduce people to each other, or as an ice-breaker.

Ad-hoc or impromptu meetings are unscheduled (or unpredictable) meetings in terms of time and place. Examples include people meeting each other by chance in the coffee-room or in the hallway. This kind of meeting, while fostering teamness and social relationships of colleagues, is totally unstructured (no agenda, no schedule, no list of participants, no expected outcome) and therefore much of the content or information is lost after a while. The conversation is also not available to team members who did not attend the meeting.

There is no expectation on the outcome of ad hoc meetings. Ad hoc meetings, however, can strengthen human bindings and foster a sense of community.

Brainstorming sessions in iBistro [Braun et al., 2001a] are held in a casual manner and might even be unscheduled. The main distinction to ad hoc and casual meetings is that they do have objectives. Therefore, on the one hand knowledge capture as well as representation of related material during such a brainstorming session is important.

Brainstorming sessions in iBistro are one type of source of information that can be later used in asynchronous collaboration.

Brainstorming sessions in iBistro can take place in a dedicated room with appropriate roomware, along with the iBistro tools, for instance meeting capturing and post-mortem structuring tools. Brainstorming sessions can also take place in *ubiquitous meetings* (derived from ubiquitous computing), where specific technical support, such as micro devices in the room, is used to support meeting participants.

Table 3.1 summarizes these three types of informal meetings and relates the type of the meeting to the meeting attributes that define the relative level of formalism. In addition to the meeting attributes that define the level of formalism of a meeting, we also distinguish the level of support for different social groups (row “Community support”) in the table.

3.1.2 PROJECT-BASED AND FUNCTIONALLY STRUCTURED ORGANIZATIONS

The classic organizational form is the *functionally structured organization*. In a functionally structured organization the staff members are grouped into departments according to speciality, such as marketing and engineering. The perceived range of a project is limited to the boundaries of the function, as each department works independently. In the *project-based organization* the functional structure still exists. The organization is built from several projects.

Matrix organizations are a combination of the classic functional and project-based structures. The matrix is built up by functional lines of authority in *vertical* direction and *horizontal* lines of project management. In weak matrices, many of the properties of the functional organization are retained and the project manager’s role is diminished. The opposite organizational model is the project-based organization with a strong role of the project manager.

	Casual Meetings in FreeWalk	Ad Hoc or Impromptu Meetings	Brainstorming Sessions in iBistro
Agenda	list of issues, prepared presentations	no	optional
Community support	same	same	balanced teams
Time	same	same	same and different
Process	low	no	none or optional
Schedule	scheduled	impromptu	optional
Location	dedicated room	anywhere, but likely in specific rooms	meeting room[1]
# of Participants	fixed	variable, small	variable, small
Relationship	casual	variable	informal
Objective[2]	low	no	yes

[1] Meetings in iBistro, either physical or virtual, take place in a specifically equipped room.
[2] Describes whether an objective is expected.

Table 3.1: Meeting attributes (shown in the left column) related to types of informal meetings.

The project manager draws on resources owned by the functional managers. This is where the vertical lines of functional management and horizontal lines of project management overlap and conflict. (Figure 4.7 in Chapter 4 displays this issue for the project in our communication survey.)

Figure 3.2 shows the project-based organizational model used in this dissertation. A project is a group of activities (Tasks) performed in a logical sequence (i.e. organized in Sub-Projects) to meet one or more basic objectives: to create change, to implement strategic plans, to fulfill contractual agreements, or to solve specified problems [SkillSoft, 2002]. A project is a temporary endeavor – it has a date of commencement and a date of completion. The end of a project is reached when the project's objectives have been reached or when it becomes obvious that the project's objectives cannot be met. In that case, the project is terminated. Each project is unique and non repetitive. During a project, limited resources from many different departments of an organization and potentially external contractors need to be coordinated. These resources include: labor and management personnel (Programmer and Executive in Figure 3.2), equipment and materials, facilities, and finances.

A group of related projects managed in a synchronized way is often called a *program*[1]. Programs are ongoing and continuous and achieve a long-term goal within an organization. Projects, in contrast, are finite.

[1]Not to be confused with a computer program.

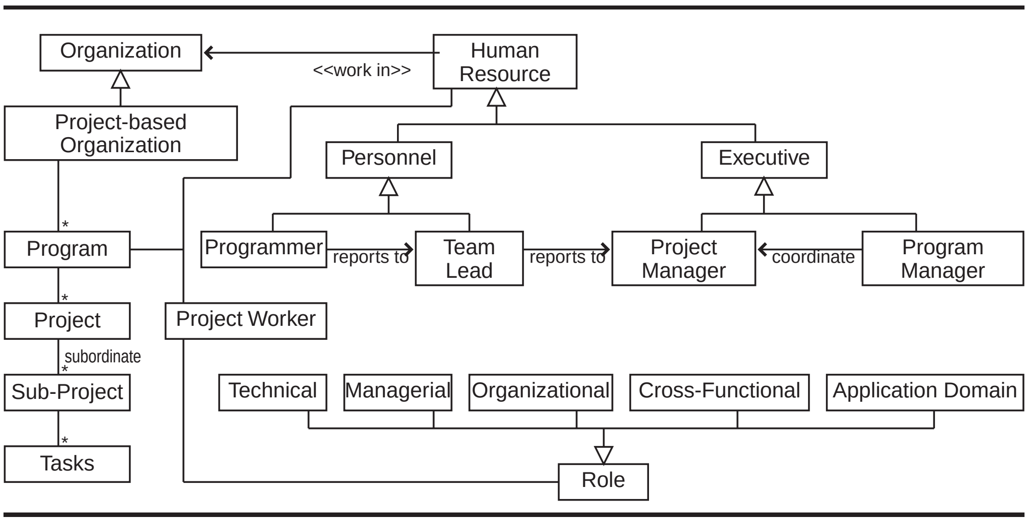


Figure 3.2: The project-based organization model used in iBistro.

Projects are performed by people. The people are working in several *roles* with according *responsibilities*. A project manager typically is the single point of responsibility. The roles and responsibilities are defined for all personnel working on the project. The roles are not static, but subject to change during the project's execution.

The role of an individual defines his or her responsibilities on the project. Each individual team member is assigned one or more *roles* and related *expectations* to be fulfilled[2]. Both are aligned with the individual's level within the hierarchy of the company or organization. A role describes the technical, functional, or managerial categories that might be assigned to a person. The expectations define the expected outcome of the work an individual performs in his or her assigned role. The expectations also define what type of tasks or issues which might occur while working on an assignment are to be addressed by the individual directly as opposed to being escalated to a higher level within the hierarchy. It is important to define and clarify the roles and expectations for all people in the project team as early as possible and revise them whenever needed. Roles and expectations define the basis for performance assessment and deliver a guideline for all personnel.

We distinguish between the types of roles in a software development project shown in Table 3.2.

[2]In practice, user specific access rights for information systems are typically assigned based on the instances of roles as shown in Table 3.2. However, in this section we focus on the managerial aspect of roles and expectations.

Category of role	Instances	Description
Management	Project manager, program manager.	Roles with direct responsibility and leadership.
Organizational	Human resources, staffing, project secretary.	Roles that administer and fulfill organizational tasks for executives without leadership and responsibility.
Technical roles	Software architecture, deployment, infrastructure, deployment, test support, migration	Technical skills and expertise.
Application domain	requirements specification and verification, system testing	Application domain-specific skills and expertise.
Cross-functional	Requirements, implementation teams, testing	Especially helper roles with lower requirement of technical skills.

Table 3.2: Roles and related instanced of roles in a software development project.

3.2 RESEARCH ISSUES

We specifically identify issues for both generic software development projects (e.g., single-site) and distributed projects. In particular, we identify three different categories of research issues:

1. *Communication issues*
2. *Organization* and *control*
3. *Knowledge & artifact management*

3.2.1 COMMUNICATION ISSUES

Communication issues includes problems related to the communication of individuals and teams over distance, time, and communities, including technical and non-technical aspects of human (project) communication.

Communication is crucial factor for project success [Curtis et al., 1988; Grinter et al., 1999; Blackburn et al., 2000; Braun et al., 2002b]. Many aspects in communication are strongly related to the dimensions of distribution defined before (Section 2.3). Communication issues get worse with increasing geographical, temporal, or community distribution [Grinter et al., 1999]. Moreover, the importance of the coherent presence of *all* categories of communication such as formal and informal is recognized [Kraut and Streeter, 1995]. Even so, an integrated approach to consider the importance of the interplay of this variety of cooperating categories and of capturing knowledge in these categories has not yet been addressed. The integration of informal-type meetings into knowledge capture and acquisition is challenging in particular, because logging a meeting will disrupt its informality.

Additionally, community distribution is an important and upcoming issue with impacts not limited to social aspects. (Community distribution also interferes with legal and organizational aspects, for instance.) Distribution is also a challenge, as personal contact is known to facilitate and ease impersonal communication, for example by lowering hurdles in initially approaching people or by helping to identify suitable contact persons [Picot et al., 1996; Pribilla et al., 1996].

Presently, no single (technical) communication infrastructure is available to incorporate means of project-wide communication with extra-project knowledge repositories of a larger organization[3]. As there is no such support tool available, the responsibility for the processing of information and knowledge contained in project communication is left to each individual user. Each user has to find a new and very individual way of information processing without making this knowledge available to colleagues. Without a common repository for organizational and project knowledge, the information remains hidden to the organization as a whole as it is embedded in scattered and unlinked emails, artifacts, documents, and individuals. With either increasing scale of the organization or increasing distribution, the problem of a missing common communication infrastructure degrades, as documents and artifacts in general are distributed.

The problem of visualizing software, for instance in models, during software development is described as one of the essential problems in software development [Brooks, 1986; Brooks, 1995]. Software invisibility (the lack of suitable tools and notation to visualize a software design graphically) is one of the essential problems in developing and understanding software. Brooks argues that (1) software structure is not embedded in a three dimensional space, resulting in lack of a natural single mapping from a conceptual design to a diagram and (2) that multiple diagrams are needed, each covering some distinct aspect.

We argue that visualization does not promise to (automatically) "find" mappings, e.g. from a design into a solution space. Instead, visualization helps programmers to find suitable mappings easier. Software diagrams and models enable negotiation and communication. Today, the Unified Modeling Language (UML, [Rumbaugh et al., 1998]) claims to solve this problem for most practical problems.

Visualization in software development projects is important if teams are distributed. Pictographic software models facilitate the communication over software design and models over distance. The UML offers a standardized notation for software model exchange. The iBistro framework and especially the iBistro

[3]Organizational knowledge repositories are, for instance, information and artifacts from related projects, a company knowledge base system, or community of interest knowledge bases.

meeting room and its roomware can be used to discuss a software design collaboratively given suitable front-end tools, for instance D-UML [Boulila et al., 2003].

3.2.2 ORGANIZATIONAL ISSUES

Organizational and managerial issues in project collaboration and control are important aspects of management which are difficult to achieve if people do not regularly meet in person and with increasing distance [Carmel, 1999; Kotlarsky, 2001].

Coordination is the ability of a project manager to direct work efforts of teams. Coordination is also the ability of a team lead to direct activities in the team, as well as it is the ability of each individual to manage his or her own workload. *Control* is necessary to assess whether measures taken are successful and how the project progresses. Coordination and control is a basis for successful project management. The dispersion of work makes it difficult to establish personal contact with individuals which eases a trustworthy assessment of the current project situation. All three dimensions of distribution (Section 2.3) complicate building up personal relationship, therefore reducing coordination and control. This issue is addressed by organizational means and special team formations, such as the "balanced team" introduced in Chapter 4. This research, however, provides a technical infrastructure which makes communication, information and knowledge, and artifacts available to teams, including managerial roles. Thus, performance and metrics can be applied.

The work in distributed environments display different human behavior compared to people who are working together physically on an every-day basis. Some authors [Grinter et al., 1999; Bruegge et al., 1999] also report a communication breakdown in distributed teams if people working together do not know each other in person. In other words, personal contact is an essential basis for electronic communication media, such as email or telephone. Loss of teamness especially affects distributed people, who do not develop or identify common attributes, which would build up a 'community'.

Advances in software development languages and tools focus primarily on improving the productivity of the individual programmer, rather than the team of developers as a whole [Booch, 2000]. Given the importance of teams to modern work life and especially distributed software development, such advances in individual productivity have "diminishing returns relative to winning the game" [Booch, 2000]. We focus on the software development team and ways to improve its productivity.

In concurrent and distributed development, different perspectives can be at different stages of elaboration and may each be subject to different development strategies. In distributed software development, this leads to projects with many actors, many representation schemes, diverse domain knowledge, and differing development strategies at different project sites. The problem of how to guide and organize development in this setting is called the *multiple perspective problem* [Easterbrook et al., 1994]. Issues related to the problem of multiple perspectives are addressed by a common ontology for global teams and by a suitable team culture (which comprehends a company jargon and terminology).

Conflicts follow an unstructured and even chaotic path. A conflict typically has two or more opposing parties who seek for public acceptance which is managed by communication activities. Mediation systems, for instance, offer a discussion forum with special support for deliberation and support the process of discussing, arguing, communicating, and negotiating issues of practical importance. Mediation systems provide particular support for trusted third parties who are responsible for moderating these kind of discussion. Features of mediation systems include issue-based discussion forums and exchange of classified documents. While conflicts in on-site projects are already challenging, global software teams need electronic support for argumentation, mediation, and negotiation due to their spatial and temporal distribution.

3.2.3 ISSUES RELATED TO KNOWLEDGE & ARTIFACT MANAGEMENT

During software development projects, many different artifacts, such as source code, (design) documents, and emails, are produced and become part of the project knowledge. Artifact management focuses on integrating those work products into a knowledge-based approach, allowing for tracking and exchange of relevant artifacts.

Tacit knowledge [Polanyi, 1966] is hard to formalize and capture in electronic repositories. While knowledge capture and acquisition always has to cover managerial and organizational means, the resulting artifacts have to be stored in ways that satisfy requirements for electronic storage systems. In other words, to store knowledge in electronic systems and later retrieve the stored information so that it is usable in a specific context, both the structure *and* the actual knowledge content have to be kept electronically.

The challenge is to find a way to chronicle or record both the structure of knowledge (given in some taxonomy or hierarchy) and the content (knowledge) itself while dealing with the mutable character of both aspects. Concretely, this means that both the content of knowledge objects and especially the structure of knowledge must be assumed to be dynamically modifiable.

This issue is addressed in the iBistro knowledge ontology and representation. We specifically suggest ontology modeling in the UML and knowledge instantiation in Java objects.

To achieve the goal of "*offering the right information to the right people at the right time*", two opposing views of the notion of knowledge can be used. The *functionalist perspective* and the *interpretivist perspective*.

The functionalist view regards knowledge as a technical object. Knowledge is considered as objects (or artifacts), such as text, notes, documents, email, source-code and so on. This kind of knowledge is easy to store and retrieve.

In the interpretivist view knowledge is seen as an inherent attribute of human beings. Accordingly, knowledge can not exist independently of human experience, social practice, and human knowledge use. Knowledge therefore is shaped by the social practices of communities of individuals. A craftsman, for instance, might have deep knowledge and skills in a specific domain, however, being unable to formalize and write down 'how to do it'. Typically, those skills are given from one foreman to his apprentice. And as the apprenticeship takes several years, this approach is obviously a long-term effort.

While typical design and requirements documentation only reflects the final decisions and hence the resulting and actually implemented artifact (software, for instance), design rationale research aims at documenting the process that led to a decision. This allows learning from earlier decisions, alternatives, and debates to improve future versions or reuse knowledge in new projects. The notion of a "design rationale" stems from the fact that design rationale were originally used during the design phase. Dutoit and Paech illustrate that principles from design rationale research also apply to other phases in the software development life cycle, specifically requirements engineering [Dutoit and Paech, 2001b]. Hence, we will use the term *rationale* to stress that rationale may be captured throughout the whole life cycle.

Rationale is still no established tool in software development, in spite of its potential importance. Engineering in general is about tradeoffs and conflict of goals. During development, engineers (and software developers) have to solve many issues during development. After finding a solution to a problem, the process of finding a solution is lost due to additional overhead, a missing tool-support for seamless integration in regular project work, and a missing method to capture rationale during the types of meetings during which they normally occur: informal meetings and brainstorming sessions.

The capture of rationale information is closely related to knowledge capturing. Rationale enable human readers to understand more about the knowledge and information provided. Explanatory statements further allow people to apply given knowledge in different situations and improve it. iBistro addresses issues resulting from unrecorded knowledge in the domain globally distributed software development. iBistro focuses on various knowledge sources[4] to capture raw

[4]The notion of a knowledge source also depicts a technical term in the context of the concept of blackboard systems in artificial intelligence. See Chapters 5.3 and 6 for more on this concept.

information/ data and knowledge, specifically in informal type of meetings, as informal communication is considered crucial for success, but difficult to capture.

Chapter 4

COMMUNICATION IN SOFTWARE DEVELOPMENT PROJECTS

"Software development used to be the domain of engineers clad in white lab coats working inside air-conditioned data centers in a handful of advanced, industrial countries. The PC revolution changed all that. In the 1990s, software development has become global."

[Erran Carmel]

The performance of individual programmers often differs by an order of magnitude [Brooks, 1986; Booch, 2000]. In the following, we argue that this is also valid for team performance, due to the typical team composition of a mix of experienced and not so experienced personnel. We found that the most effective teams showed little extra-team communication and very intense intra- team communication.

In Chapter 3, we distinguished technical and functional roles in software development teams (Table 3.2). These roles categorize teams into *technical teams* and *functional teams*. Technical teams will especially work in technical roles, and sometimes in cross-functional roles. Technical teams consist of technical experts, like programmers, while functional teams consist of application domain experts. In either category a set of tasks is assigned to team leaders. The team leaders staff their teams accordingly to the assignment and availability of people. The resulting team formation in this case is partly a random result, for instance based on availability of human resources.

Our hypothesis, then, is that a team that is composed of a balance of technical experts, application domain experts (e.g., banking, finance), and less experienced staff performs best and enables reduced communication overhead. This team composition still yields a maximum of coaching and incorporation of less experienced or weak team members. We call the resulting team formation a *balanced* *team* [Braun et al., 2002b]:

> **Hypothesis.** Balanced teams display higher team performance and greater potential for distributed work.

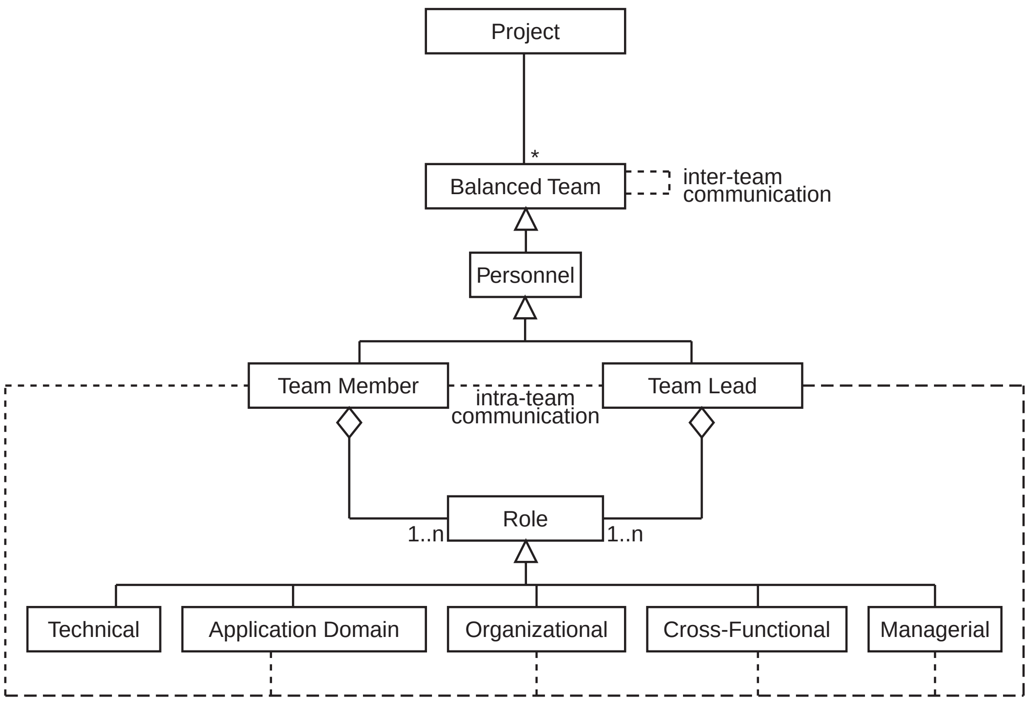


Figure 4.1: Distributed balanced teams using the iBistro infrastructure.

Figure 4.1 shows a balanced team formation. Balanced teams display higher intra-team communication and lower inter-team communication than unbalanced teams. Most of the inter-team communication is channeled through a small number of communication peers (such as the team lead, for instance). By *communication peers* we understand the parties involved in direct communication, such as the sender and the receiver of a message, or the direct counterparts in a face-to-face discussion. To enable optimal inter-team communication and coaching, the balanced team itself is a local team of co-located team members.

> **Definition.** A *balanced team* consists of co-located experts for technical and application-domain related issues along with a team lead who serves as a primary contact for internal communication as well as the primary peer for external communication. The communication metrics for a balanced team display higher intra-team communication and lower extra-team communication compared to typical team compositions.

Given a project with n persons, there are up to $\frac{n(n-1)}{2}$ potential communication interfaces (or pairs) – as any person could address anybody else on the project. In this communication scenario, all communication paths are treated equally. This is not valid for distributed projects. In distributed project scenarios, communication that extravagates one or more of the three dimensions of distribution (Section 2.3) is more "expensive" than local or internal communication. The resulting communication effort has to consider the cost of each communication path. This results in a matrix that lists whether a communication path between two nodes (team members) exists, and what the cost of this path is.

The balanced team addresses this issue by providing pre-defined communication paths between two distributed balanced teams. This means that communication between two balanced teams is channeled through the team leads or specific stakeholders in the teams. Software projects include challenges from the technical side (e.g., programming or computer science related issues) as well as in the application domain (e.g., banking or finance). In co-located teams in XP the client has to be present at the project site. In the balanced team, in contrast, the team experts for each domain of expertise act as *stakeholders* for technical or application domain issues. The stakeholders act as primary contact for all questions in their domain. If specific issues can not be solved by the team expert, this expert also acts as single point of contact for extra-team communication for remote stakeholders, including a remote client expert. The main advantage anticipated by this procedure is, beyond the improved relationship between the stakeholders due to scarce change of stakeholders, that experts rephrase the problem and gain quicker agreement with expert members of the other team. Further, communication overhead is reduced, by not simply allowing everybody to contact anybody else.

The balanced team approach addresses the "productivity paradox" observed by Blackburn [Blackburn et al., 2000]:

- almost every project manager names "best people" as the most influential single factor for project success, while best people are hard to find;
- faster firms tend to have smaller teams (except for requirements);
- more people often decrease performance on a project.

These observations are made by several authors, for instance [DeMarco and Lister, 1979; Brooks, 1995; Booch, 2000]. In a balanced team, hence, the very good people are identified based on their performance assessment from previous projects and skills list. They are distributed evenly among the project teams. This prevents, however, that one single "dream team" outperforms other teams and at the same time allows for the coaching of new and less experienced team members. Given a company culture that prevents information hiding, for instance by fostering intra-team coaching and developing related skills, this team formation can be used for teaching less experienced people by more experienced team members.

4.1 RESEARCH CONTEXT

4.1.1 IBISTRO

iBistro is an exemplary prototype for a distributed intelligent meeting room. iBistro is *distributed*, so that several local iBistros are interconnected and draw on shared resources. iBistro is an *intelligent* workplace by offering automated services to local or remote users, such as suggestions and (partly) automated acquisition of meeting knowledge. Finally, iBistro provides a dedicated *meeting room* with meeting-support and roomware.

Local iBistro's at each project site are connected over the Internet. Five fundamental concepts in iBistro support distributed informal meetings: knowledge capture, storage, knowledge representation, knowledge acquisition, and knowledge retrieval. iBistro had been developed with an empirical approach in several iterative cycles. The tool support for informal meetings were developed and evaluated at university. Because the student case studies at university are different from professional development teams, the requirements elicitation for team support were done in a large industry software development project implemented by Accenture.

4.1.2 ACADEMIC PROJECTS

We evaluated the tool-support for informal meetings in university projects. To evaluate knowledge capture and acquisition in informal meetings, we choose weekly status reviews held during a diploma or bachelor thesis and a three-month system development project. Such reviews are held between the supervisor and the student who is pursuing the thesis. In these case studies we focus on informal meetings and brainstorming sessions which take place between several students and researchers.

Globally distributed software development is evaluated during a distributed diploma thesis, done in Singapore and Munich between August and December 2002.

4.1.3 INDUSTRIAL PROJECTS

To identify actual project needs, requirements, and user acceptance criteria, we used the implementation phase of a very large banking project performed by a consultancy as source of information during requirements. We specifically analyzed project communication and project metrics [Braun et al., 2002b]. In the project under investigation, more than two hundred professionals from the consulting company, the banking corporation's IT subsidiary, the bank worked together at two project sites (both in Germany). The team members, together with banking specialists and freelancers, specified, designed, and implemented a new Enterprise Java Beans (EJB) based system to support branch office personal in real estate finance.

In the following sections in this chapter, we introduce the project under investigation and detail our findings used for iBistro requirements elicitation.

4.2 EXPERIMENTAL ENVIRONMENT

We introduce seven aspects that were characteristic for the project (see also Figure 4.2): the engagement, the consultancy, the project itself, release planning, the common technical architecture, common project metrics, the release under investigation, and finally, the team organizational structure.

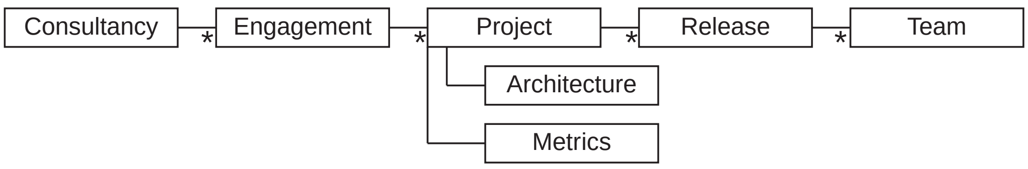


Figure 4.2: Overview of the project under investigation.

In the following sections, we instantiate these aspects for our communication survey.

4.2.1 THE CUSTOMER: A BANKING CORPORATION

The customer is one of the largest banking corporations and real estate financing companies in Europe. The bank had approximately more than 1,000 branches with over 65,000 employees at the time the survey was conducted.

The bank's goal was to redesign its real-estate business processes and software. This goal was based on strategic considerations with far reaching effects for this bank, as it specifically has a special focus on real estate financing. As a result, the project heavily affected the banking corporation's core business processes. This change had also some technical implications, managerial challenges, and additionally puts some risk on continuous business. To deal with such a fundamental change, the overall project was divided into a number of releases, each of which implemented a certain amount of functionality or modules, without affecting the working legacy system (incremental approach). The bank has both an outsourced IT operational service as well as an IT systems engineering and software development subsidiary. Nevertheless, the bank decided to give the lead and the realization of the reengineered real estate financing system to an external consultancy.

4.2.2 THE CONSULTANCY: ACCENTURE

The assignment for the consultants included the following tasks: Defining the scope of the system, requirements engineering and business process reengineering, development and ongoing maintenance of a common software architecture. Definition of the functional design and implement and test the new system. Project management wise, the responsibilities included team leadership for developers and testing teams which are composed of client and external personal, freelancers, and people from subcontractors. Finally, the system rollout planning and initial support during the rollout phase are to be managed and accomplished.

Consultants started early to analyze the specific functional and technical aspects of the banking post-merger project. A small team started defining and specifying aspects of the technical architecture by leveraging off similar experiences in the organization. Stakeholders and technical architects are found using KX, Accenture's "Knowledge eXchange" system [Brody et al., 1999]. At project start, experienced and specialized technical architects are staffed to the project to support the local teams at project initiation.

Most of the project had been conducted at two client sites in Munich and Hamburg, including requirements engineering, design, implementation, and rollout. This approach enabled continuous interaction with the client but is costly and put additional challenges to the client's infrastructure with respect to office space, technical equipment, and other resources.

4.2.3 THE PROJECT: A POST-MERGER PROJECT

The bank merged with another large banking corporation some time ago. In addition, another competitor had been acquired. In the project the following objectives had to be achieved: backoffice credit processes had to be shortened and integrated into an user-friendly system with reduced paper work and improved risk management (e.g., incorporate risks in prices of banking products). Other goals included improved sales hit rate and cost cut and increased productivity.

Moreover, the new system shall be the technical basis to integrate external sales channel, such as resellers or internet portals, into the system. The system also has to cope with some new or improved legal aspects of real estate financing that were or still are to be introduced by the European Union (for instance client assessment according to the Basel-II [BIS, 2003] rating standards).

To handle the risks of such a complex change, the new system was designed to be implemented and introduced step by step in several releases. Each release focused on a specific functionality of the overall system. The releases were designed to work as independently as possible from the final system. They are also designed to work together with the legacy system. However, some of the releases are known to be coupled tightly (i.e., sharing the same data sets or host modules, depending on data from each other). To handle this, some releases were planned to be realized concurrently. (See Figure 4.3.)

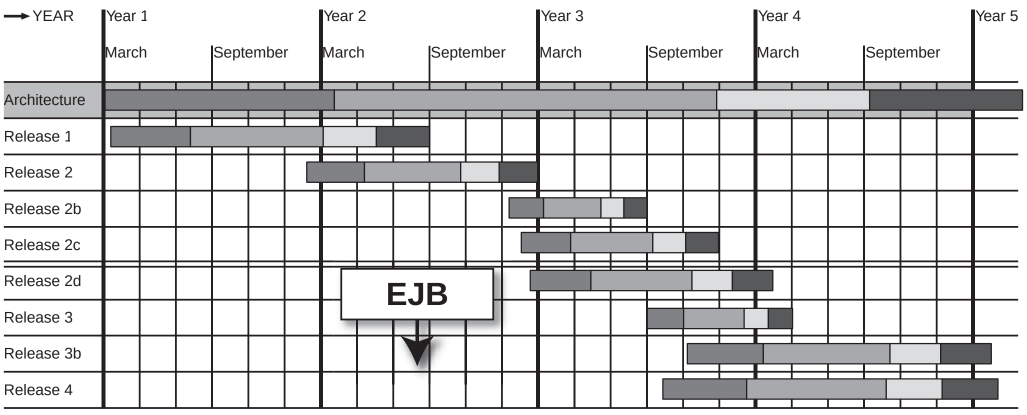


Figure 4.3: Release planning (retrospective view).

The bank decided at the end of Release 2c to change the server architecture from Microsoft COM to Sun's Enterprise Java Beans (EJB) architecture. This redirection heavily affected the architecture team and the release planning. The survey focuses on Release number 4 (the last line in Figure 4.3). This release was already developed using the new Java-based architecture.

4.2.4 TECHNICAL ARCHITECTURE AND FRAMEWORK

Part of the project was to establish a common technical architecture as a consolidated basis for all parts of the system. Even though the architecture changed from Microsoft COM to Java Enterprise Beans during the project, the comparison of teams working in an single release is made easier, as every developer has to face comparable technical challenges and uses the services and advice of the same architecture team. These preconditions are critical to compare the team performances.

The latest version of the technical infrastructure served as a framework for the application developers. Therefore, at least from the technical point of view, all developers faced the same challenges and had to implement the same type of code.

The basic data set for the window was defined in the eXtended Markup Language (XML, [W3C, 1998b]). The window specific XML structure comprehends all potential sub data sets which might be needed in that specific part of the application or business logic. The business logic and dynamic behavior itself was implemented in a Java Activity Bean, basically an implementation provided by the framework. Similar to the Activity Bean, which visually interfaces the front end, Data Beans offer basic functionality to the mainframe and host systems. Both the Activity and Data Beans were based on Enterprise Java Beans [Green, 2002].

The static part of the user interface was implemented in the eXtended Stylesheet Language (XSL, [W3C, 1998c]). The XSL stylesheet allowed the transformation of forms and data to either HTML to be displayed in the web-browser, or, for instance, Portable Document Format (PDF) for document printing or archiving.

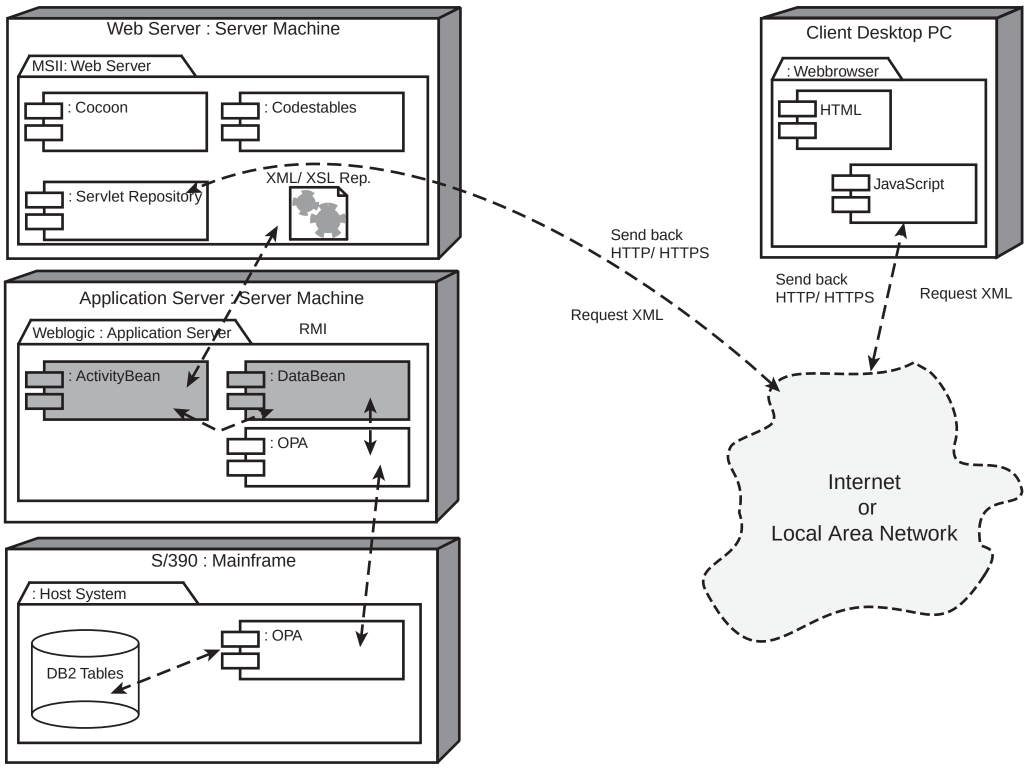
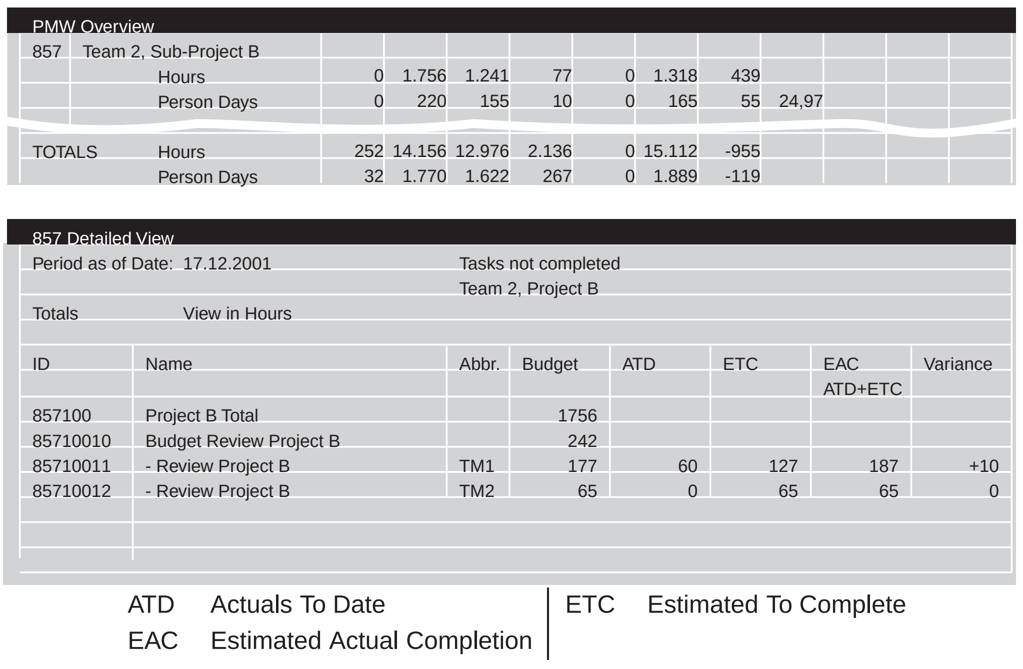


Figure 4.4: The common technical architecture used during the project. The implementation teams under investigation in this survey developed ActivityBeans and DataBeans shown in grey in the figure. The framework established a common basis for all implementation teams, resulting in comparable technical challenges. Therefore the results of the teams as well as performance metrics were comparable.
'OPA' is a specific middleware for server to host/ mainframe peer-to-peer transaction processing. Apache's 'Cocoon' is a framework for building web publications and applications, that are based on serverside XML and XSLT technologies. Cocoon was used to merge XML data and XSL stylesheets that built components for the frontend.

4.2.5 TEAM PERFORMANCE METRICS

The metrics used to measure the performance of the individual teams were based on project relevant data and known to the teams. Hence, team leads and team members knew in advance how the performance assessment was made.

Project relevant data is recorded in a Project Management Workbench (PMW) sheet, which is an electronic spreadsheet document. The PMW lists the following information: tasks, initially estimated total effort per task, actual invested time per task, and estimated to complete. This information is recorded per-project, sub-project, task, and individual team member. Tasks are listed accordingly to the initial specification of the system and further subdivided per-team with estimations made by the responsible team lead. Figure 4.5 shows an example PMW sheet.



PMW Overview

857 Team 2, Sub-Project B									
	Hours	0	1.756	1.241	77	0	1.318	439	
	Person Days	0	220	155	10	0	165	55	24,97
TOTALS	Hours	252	14.156	12.976	2.136	0	15.112	-955	
	Person Days	32	1.770	1.622	267	0	1.889	-119	

857 Detailed View

Period as of Date: 17.12.2001 — Tasks not completed, Team 2, Project B

Totals — View in Hours

ID	Name	Abbr.	Budget	ATD	ETC	EAC ATD+ETC	Variance
857100	Project B Total		1756				
85710010	Budget Review Project B		242				
85710011	- Review Project B	TM1	177	60	127	187	+10
85710012	- Review Project B	TM2	65	0	65	65	0

ATD	Actuals To Date	ETC	Estimated To Complete
EAC	Estimated Actual Completion		

Figure 4.5: Project Management Workbench (PMW) estimation and status tracking sheet.

The PMW overview sheet (the upper part of Figure 4.5, labeled "PMW Overview") shows a typical summary for the overall project. The individual calculations for each sub-project are made in the specific sub-project sheets (one example is given in the lower part of Figure 4.5, labeled "PMW Detailed View").

The following parameters assess the performance for an individual task: The actuals to date (ATD) denote the actual amount of time invested in a specific task. The hours estimated to complete (ETC) denotes how many hours the team lead estimates the specific task to take until full completion. The estimated actual completion (EAC) is the result from the actual amount of time already invested (ATD) plus the estimated time the task will need for full completion (ETC). The later figure can be compared with the budget, which is the initial estimation for the specific task.

By tracking all specific tasks in a sub-project accordingly to the work breakdown structure, the overall project success is visualized at any time during the project. The summary is shown in the PMW overview sheet.

The PMW sheet can also be used to compare team performance. The individual task are initially assigned to the project teams. This is achieved through negotiation in a common session with the project manager and the team leads. The goal is to find a fair and well-balanced assignment. During and especially after the project, the estimates for the individual assignments were compared to the EAC at the time the project ends. Assuming that all initial estimates of the budget per task do have the same quality of estimation, the comparison of budget and EAC at completion time display the team performance.

Project Management Workbenches (PMW) have been used during the whole project, including Release 4, to measure and assess team performance, project progress, and status. The initial PMW spreadsheets are built from the original estimations and the requirements documentation. The overall PMW is broken down into sub-projects. For each sub-project, the PMW lists basic and recurring tasks, such as detail design, implementation, unit and assembly testing, and so on. These basic tasks occur in every component in the release. Each member of the project team fills in his or her PMW sheet by assigning the time spent on the specific task to the corresponding PMW entries. Thus, team leads and project management knows how many time is spent and, after the task is completely finished, what the status of the overall project is.

The PMW spreadsheets are set-up in advance and are the same for every project team. Beyond this, every team implemented more than one sub-project. We therefore use the PMW for team-performance assessment in this survey.

4.2.6 THE RELEASE UNDER INVESTIGATION: RELEASE 4

The scope of Release 4 was to grasp and assess securities for the loan, to collect relevant information and properties of the real estate to be financed, and finally to calculate the costs of financing, such as charged interest, monthly rates, or taxes.

The requirements for Release 4 were defined and specified in a document, called “*Fachkonzept*”. The Fachkonzept spans more than a requirements analysis document. It also includes a top level design and the key architecture and design decisions.

Some ill-defined aspects of real estate finance strategy and some legal issues remained open after the project kick off. As a result, the project from the start required much interaction with domain experts from the client side during the project life cycle. This led to an interactive development process as illustrated in Figure 4.6. This study specifically focuses on the following three iterations in Release 4: Implementation and Unit Testing, Assembly Test, and System Test. Initial planning and deployment are introduced for completeness and introductional purposes for the other iterations.

During the initial planning and requirements engineering phase, a basic prototype was developed to demonstrate usage and flow of events of the system. This prototype was also used to check for completeness of data for the business processes and the workflow. The prototype, along with the Fachkonzept, was part of the contract and mandatory for all developers. For instance, information given in the Fachkonzept was used for test planning and to write testscripts.

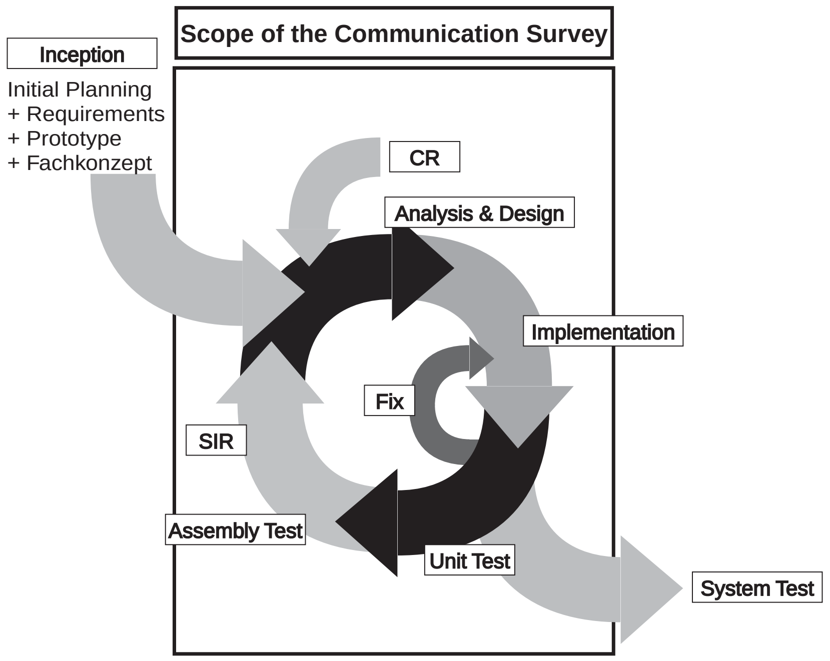


Figure 4.6: Life cycle for Release 4.

As a non-linear process was used in the project, many errors in requirements and in the Fachkonzept were found during implementation, assembly testing, and during system testing. Errors in code, design, requirements, and usability had to be fixed iteratively in the Fachkonzept, in the designs, and in the code. Additionally, many Change Requests (CR) to the Fachkonzept occurred during the iterations. A Change Request is a change to the actual specifications made in the signed-off concept and is therefore a change with impact on the contract. A SIR

(System Investigation Request) is an unexpected behavior of the system deviating from specification made in the Fachkonzept, either functionally or technically. A SIR is entered to a specific SIR-database and assigned to a team or individual for investigation. The SIR then might be considered a bug or limitation, which requires fixing of the code. A SIR can also be a functional deficiency, that is that the bug is in the Fachkonzept (in which case the Fachkonzept needs to be corrected and the code re-written or fixed accordingly.) A SIR can also have many technical reasons (such as incorrect test data, dependencies, ...). Finally, the SIR can be rejected.

4.2.7 THE DEVELOPMENT TEAMS

Figure 4.7 displays the team structure and organization used during Release 4. Functionally-structured positions are shown in vertical direction, the project-based organization is shown in horizontal direction.

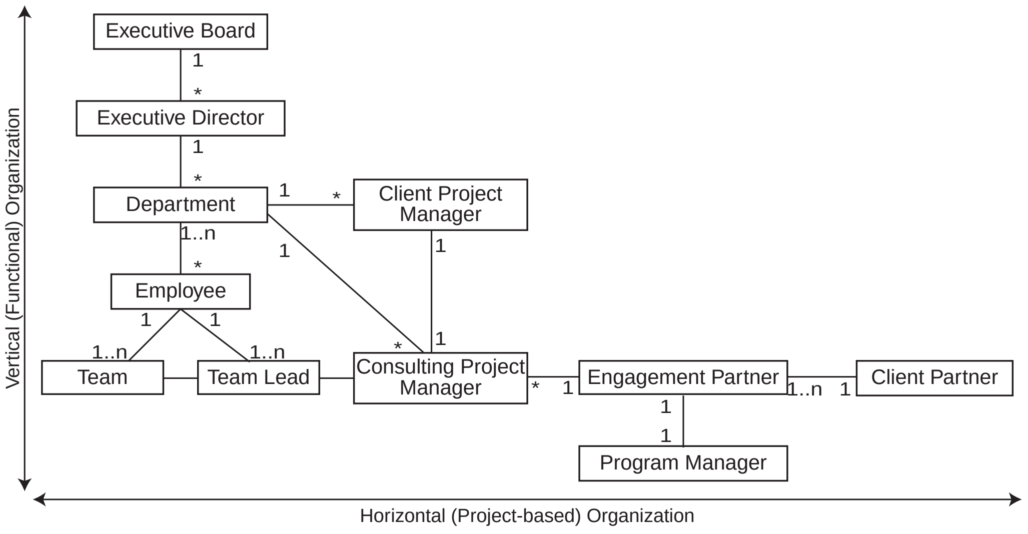


Figure 4.7: Teamstructure and Organization Chart for Release 4.

On the consulting side, the following roles were important:

The *client partner* was responsible for socializing and establishing contact to the higher level management of the client corporation, e.g., the members of the board. Typically, the client partner together with responsible board members defines the strategic need and proposes a draft for a solution. The client partner is often involved in the early proposal phases of a new project, but is not directly involved in later stages. The client partner is the topmost executive in the hierarchy, however less occupied with the actual project after inception.

The *engagement partner* is responsible for the integration of various projects that together strive to achieve a common goal in terms of a business case. The engagement partner is typically the lead executive for several projects at one client, while the client partner is responsible for more than one client.

The *program manager* directed several projects to achieve one common strategic goal or solve a business case. A program consists of several projects which consist of several releases. The program manager is therefore responsible to coordinate each project's managers to achieve the common goal. While the program manager is more on the strategic side, the engagement partner, who is also responsible for the integration of several projects, works more on the executive side.

The *project manager* leads one project and directly reports to the engagement partner. A project manager is responsible for all coordination and control related activities in one project and directly interacts with the team leaders.

The *team leader* runs a team, either an implementation, testing, architectural or other team that belong to a project. Team leaders are responsible for tracking progress within their team and identifying issues. They coordinate cooperation with other teams and foster team work. The team lead directly reports to the project manager.

Team-members came from the consultancy, the client side, and from third-parties:

Team members are either consultants, bank personnel, freelancers[1], or staff from sub contractors. Team members responsibilities range from detailed design, clarifying and negotiation of technical and application specific detail, implementation, testing, and rollout to organizational tasks, like status reporting, tracking and so on.

On the banking side, the following roles are of interest:

The *executive board* identifies a business need or charges an executive director to identify call for action. Eventually, the board awards the contract to a consultancy or internal department.

The *executive director* further is responsible for the project. Responsible department and project managers are assigned by an executive director.

Department managers and *project managers* lead the execution of the project. If a department hosts more than one project, project managers might be assigned by a department manager. The role of a project manager is typically not related to a company's hierarchy (it is a project related position), while a department manager is. Both parties (consultancy and client) appoint their own project manager, typically with higher conventionary responsibilities at the consulting side.

[1]Freelancers were increasingly used to augment the project personnel and to add human resources to the project for a limited time frame. Freelancers act like a one-man company and work for their own account.

4.3 SETUP OF THE COMMUNICATION STUDY

The goal of this survey was to analyze the communication preferences and actual communication tool usage in relation to the actual measured performance of the teams. Hence, we have chosen a combined qualitative/ quantitative approach for this survey. Quantitative data was captured throughout the whole project for each person and task weekly in a so called *turn around* sheet. The turn around sheet listed the achievements and estimates for each individual task in the project. The resulting data on progress and effort was transferred to the PMW sheet per team. This task was typically performed by the team leader.

We have chosen the combined qualitative/ quantitative approach to validate our ‘balanced team’ hypothesis (see Page 53). As much of the relevant input from our subjects was based on their personal rating and assessment, we have chosen semi-structured qualitative interviews for data collection. The rationale for this choice were as follows:

- Our goal is to investigate project member’s assessment of their actual experiences with off-the-shelf communication tools used in one specific industry project. Thus, the study was set up retrospectively.
- Personal interviewing of all project members helps us to get a realistic and comparable assessment.
- As one of our goals is to elicit knowledge for future tool support, we are also interested in unforeseen ideas or solutions provided by interviewees. Several interviewees, for instance, used communication tools that were not provided by the project, such as messaging tools.

4.3.1 APPROACH AND DATA COLLECTION

In the qualitative part of the survey, the following topics were investigated in semi-structured interviews in six stages:

Stage 1: Communication vehicles

The goals for the first stage of the interview were:

- identify means of communication actually used at the project,
- elicit unknown, forgotten, or rarely used communication vehicles;
- assess user acceptance of communication vehicles and tools;
- assess user rating and effectiveness of communication vehicles.

We investigated the frequency of occurrence of communication vehicles used at the project. We started the interview by offering a full list of communication vehicles, ranging from electronic asynchronous means of communication (such as email), over spontaneous face-to-face meetings (such as hallway meetings) to scheduled formal meetings (status meetings, for instance). Like in every stage of the interview, the interviewees also had the possibility to name alternate ideas or add communication vehicles that were used, but missing in the initial list.

During this stage, we asked all interviewees to assess the actual versus the nominal condition of communication usage (the actual usage compared to the ideal situation), and finally rate the effectiveness of different means of communication accordingly.

Stage 2: Proportion of intra vs. extra-team communication

The primary goal of stage two was to categorize the actual interviewee by his or her communication pattern.

We studied the proportion of intra-team and extra-team communication handled by individual interviewees. This information was not treated isolated, but particularly meaningful in the context of the communication vehicles used (Stage 1), the role and responsibility of the specific interviewee (Stage 6), and finally in the facet of the interviewee's team performance (quantitative part of the survey).

Stage 3, 4: Triggers for internal (3) and external (4) communication

The goal was to compare the following *categories of triggers*:

- technical issues
- application domain specific issues
- organizational and management issues
- value-oriented topics
- and interfaces to other teams or individuals

Stages 3 and 4 compared five categories of triggers that start internal and external communication and compares the effectiveness in finding appropriate solutions and the quality of solutions found.

Stage 5: Communicating peers

The goal for stage 5 was to analyse the communication network and identify the communicating peers (the communicating persons and teams):

- **technical issues:** infrastructure, host architecture, server architecture;
- **application domain:** application domain department;

- **organizational, management, and value-oriented topics:** team leadership, project management (consulting), project management at client's IT subsidiary, program management, and engagement partner;
- **Interfaces:** other implementation teams within the same release, other implementation teams within other releases.

Stage 6: Roles & responsibilities

Stage 5 focused on the communication peers (teams or individuals within other teams). The communication peers emerge from the organizational model (shown in Figure 4.7), the communication triggers (Stages 3 and 4), and the roles of the interviewees (Stage 6). Therefore, to align the communication network both team internal and external, we had to ask for the interviewee's roles and responsibilities.

In Stage 6, we identified the interviewee's role accordingly to the organizational model:

- project management;
- team lead;
- team member.

The final stage of our survey was strongly related with earlier stages and needs to assess the interviewee's ratings accordingly to his or her role. To do so, the interviewee's answers had to consider his or her role. A team lead, for instance, typically will communicate much more with project management than team members. Team leads also display different communication patterns, as they have to attend certain formal meetings (status meetings, for instance). Moreover, people who are working in requirements engineering typically display more communication with the application domain specialists.

The following categories of responsibilities were offered:

- requirements, top level design;
- detailed design server;
- detailed design host;
- implementation server;
- implementation host;
- assembly test;
- system test preparation;
- system test support;
- fixing during system test;
- system test;
- deployment;
- architecture;
- administration;
- team leadership;
- user support;
- ordering, acquisition;
- roll-out.

4.3.2 QUANTITATIVE PERFORMANCE METRICS

Our hypothesis was that teams that display a team formation and communication patterns accordingly to a balanced team perform *better* than other teams. The qualitative part of the analysis, as described before, allowed us to elicit *how* teams actually worked together. The quantitative part of the analysis determined *who* are actually the good performers.

4.3.3 POTENTIAL RESTRAINTS TO SCIENTIFIC VALIDITY

Even though the survey was supported by executives, due to the German law the bank's works council must approve such a questionnaire, as worker's interests could potentially be affected. Apprehension in our case was, that the collected data could be used to assess an individual's performance, or gather further insights about specific behavior, attitudes, or to learn about personal communication traces (the study also asked for personal relationship of colleagues). The survey therefore did not include subcontractors and bank employees. Hence, the final sample only contained the answers of consultants and freelancers.

Consultants and freelancers are often considered to have a different work attitude than regular employees. While we do not know of any investigation to prove or disprove this rating, we believe that at least those parts of the survey that describe facts of the project communication and cooperation are not biased. However, in the following we explicitly mark passages, especially assessing statements, where it is important to note that the results do only reflect the consultant's view.

Figure 4.8 shows the demography of the interviewees. All project members, including executives, were younger than 40 years, with 29 years in an average. Moreover, the team members and programmers were at most 30 years. In contrast to a survey by Blackburn et al. [Blackburn et al., 2000], all people working in Release 4 were fluent with email and internet. All interviewees were trained in using many electronic work tools for communication and other tasks.

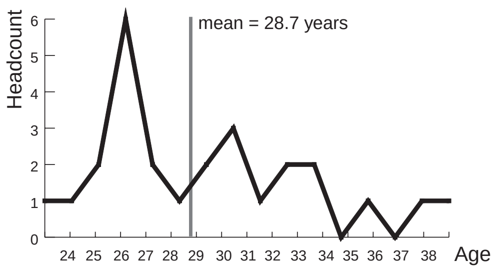


Figure 4.8: Demographic distribution and mean age of the interviewees. The average age of the participants of the study was 28.7 years with a quantum of 25 interviewees.

4.4 RESULTS

In this section we display the results to identify the "best" performing teams and the findings of the six stages of the qualitative survey.

4.4.1 TEAM PERFORMANCE

Figure 4.9 shows the comparison of the *estimated cumulative effort* (or estimation) for the tasks assigned to each individual team, which is the total budget for the team. The estimated cumulative effort is shown as a black bar in Figure 4.9). The estimation is compared with the *final actuals to date* (ATD) at completion, shown as a light-grey bar in Figure 4.9. The comparison shows how each team performed compared to the estimates and in relation to other teams.

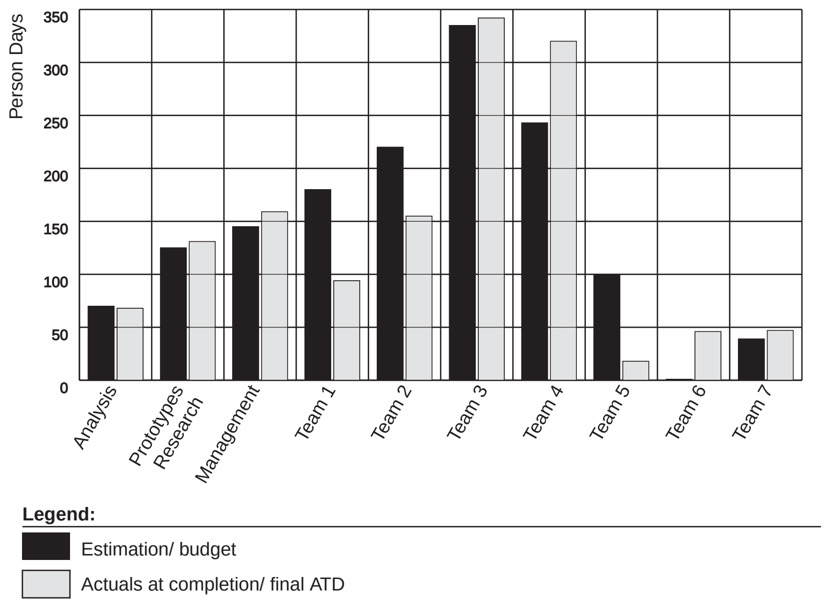


Figure 4.9: Team performance, measured by comparing the estimations (based on the work breakdown structure) and the actual effort per team in person days.

The teams demonstrated about the same performance. The actuals to date for the teams were actually close to the estimates. The individual performance of the teams is discussed consecutively:

Team 1 and 2: These teams only are organized accordingly to the balanced team idea. The teams display a well-balanced composition of application-domain and technical expertise, new hires and less experienced staff as well as experienced team members. Communication is fostered inside the team and much of the communication is channeled through and addressed by the team lead.

Team 3: This team performed close to the estimates and within the expectations, that included some contingency.

Team 4: This team notably underperformed. The team overall struggled with managerial issues and functional misinterpretations. As a consequence, the team composition had been changed in the mid term.

Team 5: This team started late and did not finish by the end of the survey due to missing requirements and change requests.

Team 6: This task was a change request for additional functionality. The task has not been estimated and has been charged besides the main project.

Team 7: This small context-project implemented an internet-portal for external service providers, including the necessary security architecture. The team performed close to the estimates, while still showing some delay (within contingency).

There are no estimates listed for infrastructure and technical architecture in Figure 4.9, as those tasks can hardly be compared directly to implementation teams.

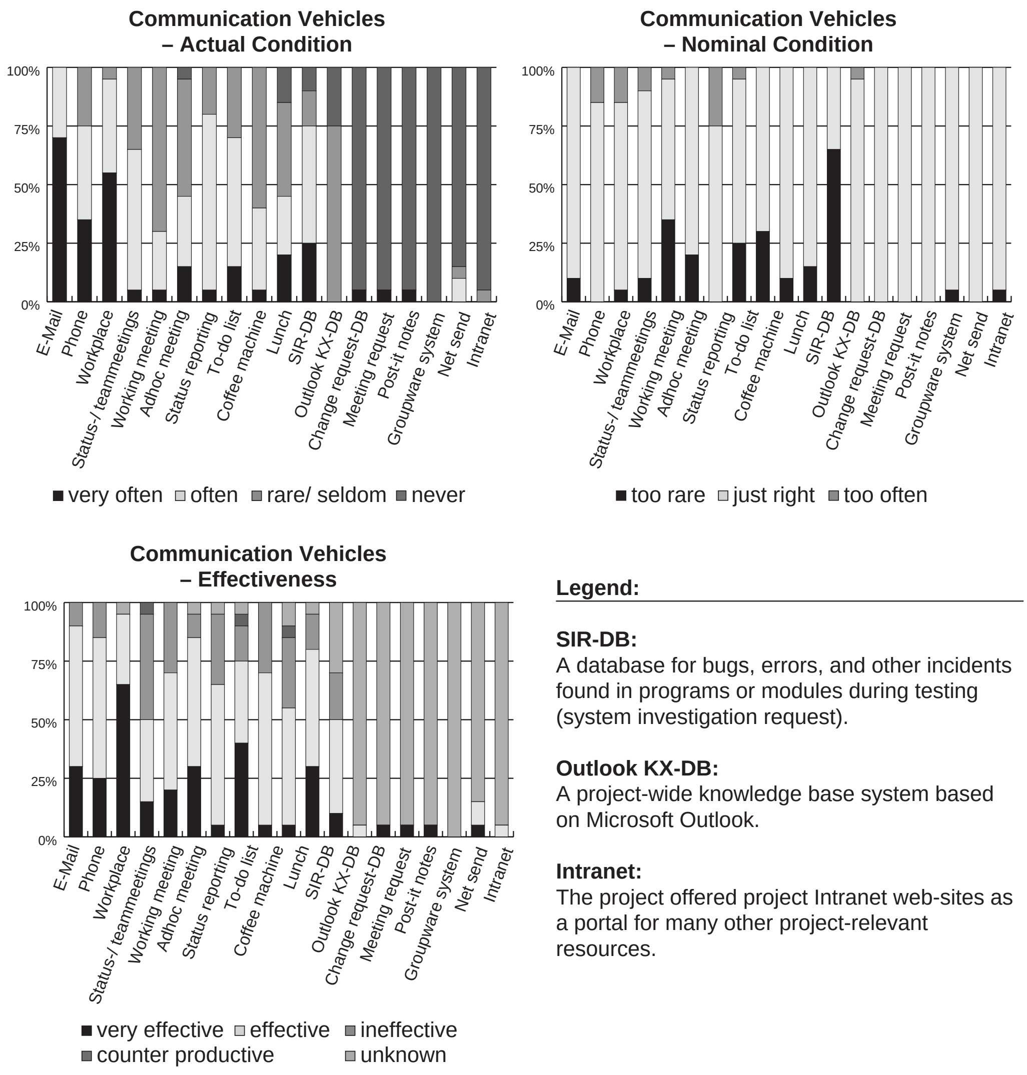
4.4.2 ELECTRONIC VS. FACE-TO-FACE COMMUNICATION

Figure 4.10 displays the usage of communication vehicles, the ideal situation, and the assessment. We analyzed the usage of all three major types of communication vehicles – electronically supported communication, face-to-face meetings, and ad-hoc meetings. The results were that all of these types of communication were used regularly, with a notably decrease from electronic document exchange down to ad-hoc meetings. Email was by far the most often used individual communication vehicle, while ad-hoc meetings, including accidental meetings at the coffee machine, occurred seldom.

Nominal condition describes how the situation as it would be in a perfect or ideal project environment accordingly to the rating of the interviewees. The nominal condition for electronic means of communication is rated as "used too seldom", especially the Knowledge Xchange (KX) [2] database. Few (1.5%) people rated phone calls to be used too often.

[2]Knowledge Xchange (KX) is an Accenture knowledge base with project and technology related company-wide information. Moreover, KX identifies stakeholders within Accenture, therefore fostering collaboration.

Effectiveness rating describes how valuable each communication media was for the project success accordingly to the interviewees. When we take a closer look on effectiveness rating, we find that electronic means of communication and electronically exchanged documents are rated “very effective” and “effective”, while the effectiveness rating for in-person meetings was slightly worse, and ad-hoc meetings were rated notable worse.



Legend:

SIR-DB:
A database for bugs, errors, and other incidents found in programs or modules during testing (system investigation request).

Outlook KX-DB:
A project-wide knowledge base system based on Microsoft Outlook.

Intranet:
The project offered project Intranet web-sites as a portal for many other project-relevant resources.

Figure 4.10: Overview of actual condition, nominal condition, and effectiveness of communication vehicles used.

4.4.3 INTRA-TEAM VERSUS INTER-TEAM COMMUNICATION

The majority of consultants working at the project (45%) had most of their communicating peers within their own team; about 40%, rated the intra versus inter-team communication to be distributed evenly. Finally, 15% observed an overbalance of communication toward extra-team peers. The "roles & responsibilities" part of the survey shows that the latter group of people is mainly in the role of team leads or project managers.

We then took a close look on the issues or problems that triggered either intra or extra-team communication. We found that beyond problems that were expected to be solved inside the team (for instance work assignment and organization), many issues that were typically expected to be discussed outside the team were actually addressed internally in Teams 1 and 2 (for instance technical and domain-specific issues). The reason might be that in the Balanced Teams 1 and 2 stakeholders for all kinds of problems that were typically addressed externally were available within the same team.

Overall, both the frequency of occurrence of internal communication and the quality of the solutions found surpasses the results for extra-team communication with one exception. The server architecture team at this specific project performed very well, both accordingly to the performance assessment in Stage 1 as well as due to the rating by the supported team. Interestingly, nevertheless the rating for their advice on technical issues was rated similar, in fact slightly worse, than the quality of solution found internally.

Figure 4.11 shows a side-by-side aggregated comparison of intra versus extra-team communication for all groups.

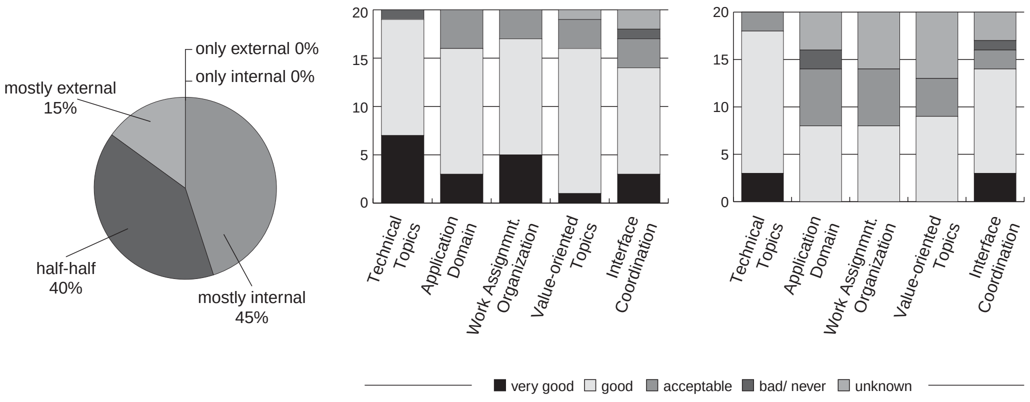


Figure 4.11: Quality of intra versus extra-team communication. The graph on the left shows the percentage of extra versus intra-team communication. The graph in the middle shows the rating of the quality and effectiveness found internally. The graph on the right displays the rating for solutions found through external communication.

4.4.4 COMMUNICATION PEERS

The communication peers addressed in extra-team communication reflect the roles of the interviewees. This results in an even distribution of communication peers according to their responsibilities. Hence, we focus on a smaller selection of peers which are more meaningful: the infrastructure (technical equipment and related services), mainframe architecture, server architecture, and upper management (project and program management, engagement partner). First, we focus on the "bad performers" in this part of the survey, namely the infrastructure and mainframe architecture teams. In both cases, communication efficiency with those teams is rated ineffective or even counterproductive in terms of finding a suitable solution in 50% or more of the answers. The interviewees reported, that as a result, people tried to avoid communicating with those teams and tried to compensate this either internally or by finding experts in other teams.

In contrast, the server architecture team was rated "very effective" and "effective" in terms of finding a solution in more than 94% of all answers. Upper management also received ratings above the average. This might be also a result of the continuous personal availability of managers at the project location. However, regular personal contact with executives is considered a crucial and motivating aspect in the interviewee's worklife.

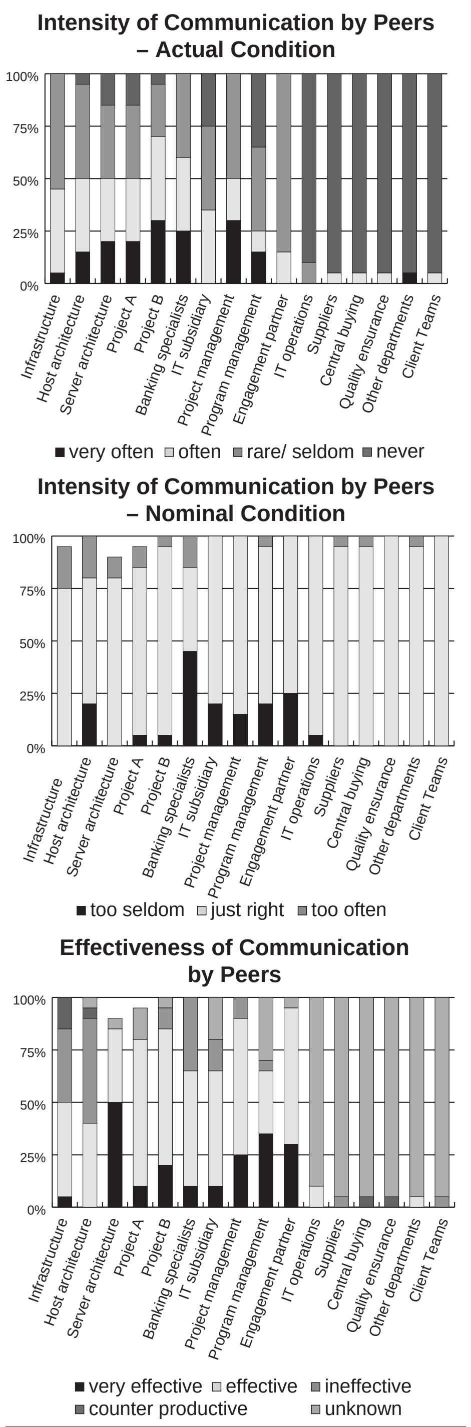


Figure 4.12: Communicating Peers.

4.4.5 ROLES AND RESPONSIBILITIES

Most of the interviewees were members of a team with no management or team leader responsibilities (76%), 16% were team leaders, and 8% were project managers or in a management-related role. The detailed view of responsibilities helps us to deskew the answers given in the communication peers stage. Most interviewees addressed communication peers accordingly to their role, especially if those peers were located outside their team. Respectively, most of the extra-team communication was performed by team leads. Team leads of the highly effective Teams 1 and 2 established contact to extra-team peers. While all team members typically had multiple responsibilities, especially teams that performed better than the average showed comparable communication patterns. This was a result of well-defined assignment of responsibilities and related communication peers.

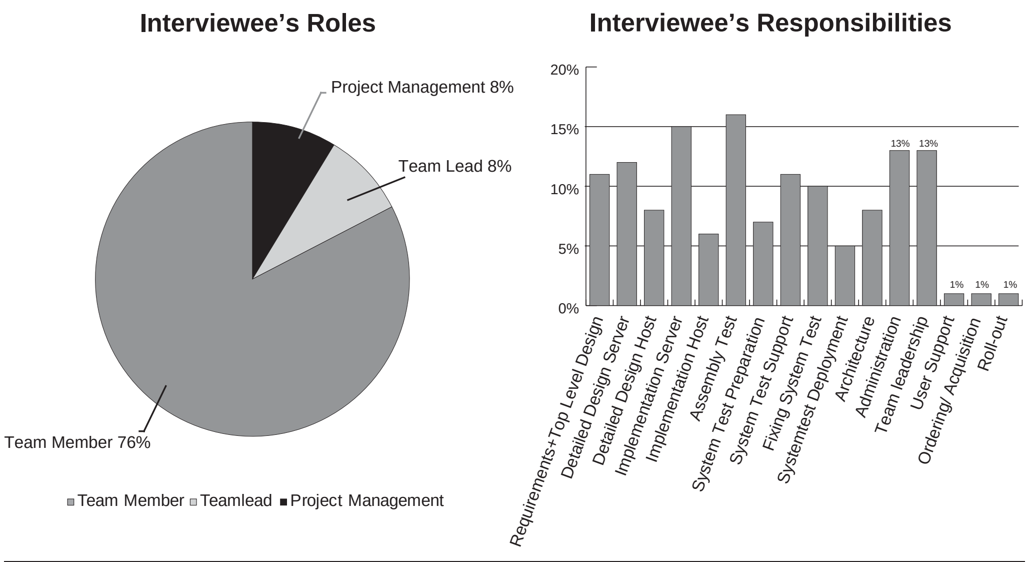


Figure 4.13: Roles and Responsibilities

Typically, external communication occurs accordingly to the roles, instead of the level within the hierarchy. For Teams 1 and 2, much external communication had been addressed by the team leads after rephrasing the questions inside the team. Other teams were seeking advice outside the team regularly. Especially members of the least effective Team 4 were known for contacting the architecture team very often, instead of first addressing technical questions inside the team. This created a disbalance of high external versus low internal communication.

4.5 SUMMARY OF RESULTS

Team-communication was strongly related to team performance. The best performing Teams 1 and 2 displayed much more internal than external communication. The worst performing Team 4, in contrast, communicated more externally than internally. We further found that the two extremes – Team 1/2 vs. Team 4 – differed especially in the team composition: both teams had experienced and less experienced people. However, while Teams 1 and 2 named dedicated people responsible for technical and functional issues to serve as a single point of contact for internal as well as for external communication, Team 4 did not make such a decision. This directly resulted in different team communication patterns and performance. In the Balanced Teams 1 and 2, different communication peers were addressed by the same people within the team (for instance, technical vs. application domain) and the amount of extra-team communication is reduced tremendously (Figure 4.11). The balanced teams displayed a behavior that requires minimal extra-team communication while yielding best performance. In the non-balanced teams (Teams 3, 4, 5, and 7 in Figure 4.9), team members had to contact external peers for those domains of expertise with no stakeholder available in their team. This fact had also been reported as a problem specifically by the architecture team, as technical questions put a huge workload on them.

The data also showed that electronic forms of communication, especially computer-based, are widely accepted by users and rated very efficient (Figure 4.10). Informal face-to-face meetings are still very important and known to be a crucial factor in setting a suitable basis for later electronic communication [Kraut and Streeter, 1995; Grinter et al., 1999]. However, due to reduced extra-team communication, a project with mostly balanced teams could be distributed without the usual penalties incurred by distributed projects due to the organization of extra-team communication through specific peers.

The results display a high acceptance and even preference of email and face-to-face meetings. The effectiveness of meeting face-to-face in the workplace is rated specifically high. There is no resentment or disfavor of electronic communication, however, the importance of personal meetings is very high. The importance of ad-hoc meetings, however, is rated relatively low.

Teams 1 and 2 display strong inter-team communication for the application-domain as well as technical issues. Teams 1 and 2 performed extra-team communication by-level (team lead) instead of by-role. If the external communication was performed by a single (dedicated) person, the request first generates internal discussion. The process of rephrasing the question and trying to understand its impact involved discussion and further research (for instance to look up technical terms or terminology). This at times already leads to a solution or at least eases further steps in finding a resolution. Moreover, the output of this discussion was documented, as it was written text (in a document or email).

The assessment that a more formal process supports quality and improves effectiveness is supported by a study performed during requirements engineering by Damian et al. [Damian et al., 2000].

4.6 RELATED STUDIES AND COMPARISON

4.6.1 OVERVIEW OF RELATED STUDIES

Blackburn, Scudder, and Wassenhove [Blackburn et al., 2000] started interviewing managers at hardware and software developers in Europe in 1992 to gain insight into the management practices that best support shorter development cycles and greater productivity and the differences between hardware and software development. Later, the empirical study was extended to projects in Japan and the United States. The study specifically investigated certain project management factors that reduce overall software development time. Eleven factors were chosen and rated accordingly to a 1-5 Likert[3] scale. These factors were: The use of prototyping, better initial customer specifications, the use of CASE tools and technology, concurrent development, less rework, improved project management, better testing strategies, reuse, smaller modules/ units, improved team communication, and finally better people.

In summary, Blackburn et al. rated people and communication as the most important factors, while smaller modules/ units, CASE tools, and prototyping were rated with much less relative importance. Blackburn et al., in contrast to our survey, also found low acceptance of electronic communication media and electronic support in general.

Curtis et al. [Curtis et al., 1988], in a field study of several large projects, observed that documentation does not reduce the need for communication, in particular, during the early phases of the project, when stakeholders coordinate their representational conventions, and create informal communication networks. Curtis et al. also observed that obstacles in informal communication (e.g., organizational barriers and geographical distance) can lead to misunderstandings in design conventions and rationale.

Kraut and Streeter [Kraut and Streeter, 1995] noted that formal communication (e.g., structured meeting, formal specifications, inspections) is useful for routine coordination while informal communication (e.g., hallway conversations, telephone calls, brainstorming) is needed in the face of uncertainty and unanticipated problems, which are typical for software development. In their study, they observed that the need for informal communication increases dra-

[3]An unidimensional scaling method to related pre-defined values.

matically as the size and complexity of the software increases.

Grinter et al. [Grinter et al., 1999] focused on distributed projects. They studied several cases using different organizational models for coordination. They confirmed the findings of Kraut and Streeter about breakdowns in informal communication in the distributed case. In addition, they found that unequal distribution of project mass around sites and the difficulty in finding experts were recurring issues, independent of project organization.

Dutoit and Bruegge [Dutoit and Bruegge, 1998] studied the relationship between communication and performance in team-based projects. They related metrics on communication media with metrics on products and observed changes in communication can have a significant impact on outcome. Some evidence also indicated that an excess of inter-team communication (in relation to intra-team communication) is a symptom of poor team performance.

Seaman and Basili [Seaman and Basili, 1998] studied the impact of organizational roles on communication and performance, in particular during inspections. They note that organizational barriers (e.g., the presence of a manager during an inspection) can degrade communication and result in a degraded outcome.

Damian, Eberlein, Shaw, and Gaines [Damian et al., 2000] challenge the claims of traditional media-effect theories. They present their findings that, contrary to traditional wisdom, teams using face-to-face meetings in requirements negotiation do not perform any better than those using video conferencing and computer-based communication. The study evaluates four distributed group configurations using several compositions of facilitators, customers, and system analysts. The analysis and comparison of the team performance do not support traditional claims that those groups using the richest communication media perform any better than those using leaner media.

4.6.2 RESULTS COMPARED TO RELATED STUDIES

In contrast to Blackburn, Scudder, and Wassenhove [Blackburn et al., 2000], electronic communication and related tools clearly experienced the highest ratings for both user acceptance and effectiveness assessments in our survey. We further do not unrestrictedly support Blackburn et al.'s conclusion that 'best people' within a team are the most influential factor for project success. The main differences, in detail, are the roles of the interviewees and the age of both the interviewees and electronic tools used. While Blackburn et al. exclusively considered executives (project managers) in their survey, we asked all team members (executives, team leaders, and team members/ programmers) in the project. The second differing aspect considers the age of interviewees and the age of tools used. In our survey, the interviewees were clearly younger than 30 years in an average.

(The mean age was 28.7 years. Figure 4.8 displays the demographic distribution of *all* project members, including executives.) The younger people are typically used to electronic communication tools from university or private life. Blackburn et al.'s study suggests itself that the interviewees were older, as all of them were in an executive position. Regarding the age of tools used, in our project situation all electronic tools were introduced some years before, but are updated and improved regularly. Blackburn et al.'s study was conducted 10 years earlier (1992). This suggests that the tools available at that time did not meet the user's requirements as one would expect ten years after. Moreover, the (elder) interviewees obviously were not used to computers, email, and internet as younger users today are. This led to user resistance in Blackburn et al.'s study.

Finally, we experienced that the increased effort from rephrasing communication to be done by team leads in Teams 1 and 2 improved communication in general and moreover made the process and its outcome available for later reuse. This experience is in accordance with the study done in requirements engineering by Damian et al [Damian et al., 2000].

The teams under investigation worked in a non-distributed setting in a large organization. However, the organization of balanced teams in this experimental setup allows us to transfer part of the encountered advantages, such as organized team-external communication, to distributed projects. In turn, this suggests that balanced teams are also well-prepared to tackle distribution issues and scenarios. We will discuss this in more detail in the next chapter. iBistro, hence, should in particular address the specific need of face-to-face conversation of project members, for instance contact to upper management, counseling, and communication peers in other teams.

CHAPTER 5

THE IBISTRO SYSTEM

"The problem with WYSIWYG is that what you see is also all you've got."

Attributed to
[BRIAN REID] *and/or* [BRIAN KERNIGHAN]

In this chapter, we describe features of iBistro, an augmented and informal meeting space for distributed teams. It serves as our testbed for experimentation with distributed software teams. iBistro allows to evaluate related ideas for the support of distributed balanced teams in software development.

We start by illustrating several visionary scenarios with two development sites at Singapore and Munich. We then introduce several levels of distribution and relate them to those scenarios. Finally, we detail three representative scenarios with distribution according to time, place, and community.

5.1 VISIONARY SCENARIOS

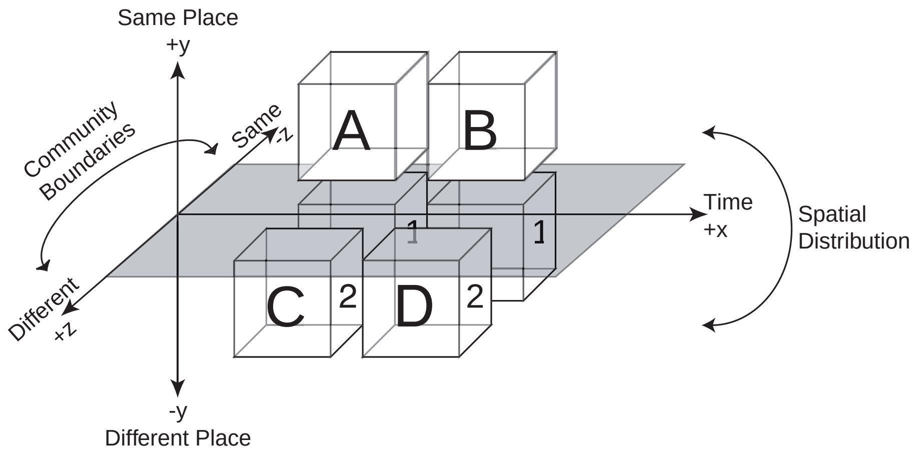
For the following description we assume a small project with two globally distributed teams. We further assume that teams, either in single-site or the distributed scenarios, are organized as balanced teams from a small group of people with complementary skills and roles. Every person is assigned several tasks accordingly to his or her role within the team. Most tasks, including organizational and managerial tasks, produce a work product. The roles and expectations, tasks, work breakdown structure are embedded in documents and spreadsheets. Artifacts of development are for instance UML diagrams and files stored under version control. By using iBistro's services as described in the following scenarios, the stored information and artifacts become the group memory and knowledge for the balanced teams.

In the following sections we use the example of a distributed scenario between the National University of Singapore (NUS) and Technische Universität München (TUM) for illustrative purposes. This example is chosen because many insights can be taken directly from our testbed between NUS and TUM in Fall and Winter 2002. It does not mean, however, that the number of sites in iBistro is limited to two sites.

5.2 IBISTRO SCENARIOS

Figure 5.1 shows the three dimensions of distribution (introduced in Section 2.3): The *x-axis* denotes different time (distribution in time), the *y-axis* shows same place (on the positive axis) versus different place (on the negative axis) scenarios. Different community scenarios are designated on the positive *z-axis*, while same community scenarios are shown on the negative z-axis.

The first example scenario is a single-site, same place/ same time informal meeting held in Munich (Cell A in Figure 5.1). After that, we depict how the content of a meeting held in Munich is used in Singapore a few days later. This scenario also reflects aspect of different communities (Cell D_1). Our final scenario (see Cell C_2) illustrates a real-time synchronous meeting between Singapore and Munich. In this scenario the live-support of different communities is our main concern. We set aside, however, the many technical and HCI challenges which would occur in the scenario, accordingly.



Cube	Scenario	Time	Place	Community	Scenario
A	A single-site meeting in Munich	same	same	same	Section 5.2.1
B	Resumption of a meeting in the same meeting room in Munich.	different	same	same	–
C_1	A live meeting between two sites in Munich. Meetings located in different places are considered to be in the same community in this scenario, if a meeting is located in two office buildings within the same town.	same	different	same	–
C_2	A live meeting distributed between Munich and Singapore Meetings located in different places are considered in different communities for the scenario, if a meeting is distributed over two continents.	same	different	different	Section 5.2.3
D_1	Resumption of a previous meeting at a different site within the same town.	different	different	same	–
D_2	In this scenario, a meeting held previously in Munich is continued later respectively used as source of information for a meeting that takes place in Singapore.	different	different	different	Section 5.2.2

Figure 5.1: Overview of scenarios and their distribution over space, time, and community. The y-axis denotes the spatial distribution boundaries. Community boundaries are visualized from the rear to the front on the z-axis. Scenarios for different communities located in the same place are omitted.

5.2.1 A SINGLE-SITE MEETING IN IBISTRO

The simplest form of a meeting within iBistro is the same time/same place meeting (Cell A in Figure 5.1). Assume three meeting participants who are developing scenarios during a requirements elicitation session: Alice is a consultant responsible for system development. Bob is a user of the system to be built and a domain specialist, e.g., investment banking. Claire represents the customer of Alice's consulting firm, e.g., a bank. Figure 5.2 shows the timely occurrence of meeting events in the single-site meeting scenario.

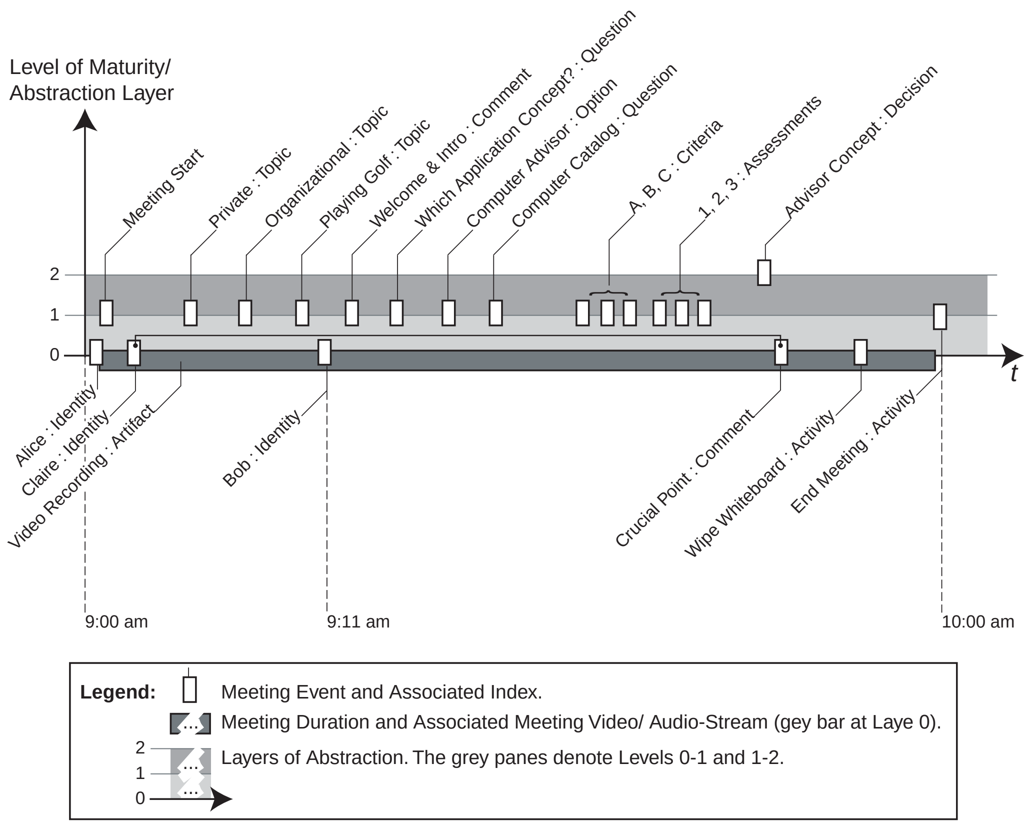

Level of Maturity/ Abstraction Layer
Meeting Start
Private : Topic
Organizational : Topic
Playing Golf : Topic
Welcome & Intro : Comment
Which Application Concept? : Question
Computer Advisor : Option
Computer Catalog : Question
A, B, C : Criteria
1, 2, 3 : Assessments
Advisor Concept : Decision
2
1
0
t
Alice : Identity
Claire : Identity
Video Recording : Artifact
Bob : Identity
Crucial Point : Comment
Wipe Whiteboard : Activity
End Meeting : Activity
9:00 am
9:11 am
10:00 am
Legend:
Meeting Event and Associated Index.
Meeting Duration and Associated Meeting Video/ Audio-Stream (gey bar at Laye 0).
Layers of Abstraction. The grey panes denote Levels 0-1 and 1-2.

Figure 5.2: Flow of events in the iBistro 'Munich' single-site meeting example. The chosen representation displays the chronological occurrence of events in a meeting. This view is therefore also called *temporal view*. (See Section 6.6 for an overview of knowledge retrieval and visualizations.)

Using electronic badges (such as the Active Badge Location System [Want et al., 1992]) given to each of the meeting participants at the entrance of the building, iBistro knows about the presence of the individual stakeholders and is able to deal with a varying number of meeting participants. This is important to allow for the assignment of the content (e.g., a single requirement) to the individual stakeholders. While sketches and notes on the electronic whiteboard can be assigned to a meeting participant automatically by assigning one specific electronic pen (with a specific color) to each participant, higher-level content, such as audio comments or video gestures, are assigned to a participant during the post-meeting process.

In our scenario, Bob is late for the meeting and arrives in iBistro after Alice and Claire have already started. Before Bob's arrival, Alice and Claire, after talking informally and privately, also talked about organizational issues, such as how to arrange the consultant's arrival at the client site and what rooms to use for software developers. However, the discussion prior to Bob's arrival also briefly touched private topics; in this case, both women play golf and shared some thoughts about their play over the weekend.

Knowledge acquisition is done by capturing the audio and video of the meeting as well as by capturing any sketches, notes, and drawings made throughout the meeting. The meeting video is shown as an artifact ("Video recording : Artifact") in Figure 5.2. The grey bar at Abstraction Layer 0 represents the video and audio file. Video and audio capture is triggered by the "Meeting Start" event and spans through the whole meeting until the "Meeting End" event stops video capturing. By allowing the manual orientation of the camera, meeting participants may use a laptop computer, paper, or even napkins to draw or to write on. For this scenario, the meeting community uses an electronic whiteboard. The whiteboards' content is captured in a separate movie by the electronic whiteboard itself in iBistro (as opposed to whiteboard-capture systems such as Zombieboard [Saund, 1998], which use static images). The history of drawings or notes is saved in a sequence of images.

Alice, Bob, and Claire are able to talk easily about the requirements of their project. They can also be sure that critical team interaction is saved in the audio/ video stream and is available for later processing and structuring. During the course of the meeting, two different application concepts are developed. The first one is a context sensitive computer advisor who guides the user by offering suggestions based on the state of their accounts with the bank. The second one is a computer catalog that enables users to browse and search through the complete range of products offered by the bank. At a critical point during the meeting, Claire, the client decides to set aside the catalog concept in favor of the advisor concept. Alice decides that this is a crucial point in this meeting and creates a comment. The whiteboard is then erased and the remainder of the meeting is dedicated to the advisor concept.

After the meeting, during the post-processing phase, Alice navigates through the meeting record, which consists of the audio/ video stream, and the set of events captured (including the whiteboard sketches).Alice navigates through the meeting along its timeline using the MEETINGGENERATOR tool. The time needed for post-processing the meeting is shortened drastically by offering any contextual event as well as subsequent changing content of the whiteboard as an index into the captured meeting. From Bob's late arrival to the meeting, for instance, Alice can easily distinguish between the strategy topics first discussed with Claire from the domain specific topics discussed after Bob's arrival. The event associated with wiping the white board is used to visualize a context switch. Hence, Alice is able to isolate the discussion associated with the two application concepts. She creates two option events representing each concept and a decision event that she associates with the video segment when Claire made her decision. During this process, Alice creates the knowledge base for that meeting by evaluating the segments of the captured videos. In contrast to reviewing the whole meeting video stream, Alice only annotates smaller chunks of the video which are indexed by the events (as shown in Figure 5.2). This approach drastically shortens the post-mortem process comparable to the amount of time which would be needed to take conventional minutes.

The post-mortem process translates lower-level captured information (such as audio and video streams and whiteboard snapshots) into higher level content (such as requirements and their rationale). One crucial point during that phase is that information interpreted that way is linked automatically with its originating source and related contextual information, e.g., a stakeholder (identity) to allow for the later sorting of knowledge by different criteria (e.g., author, time of occurrence, type of event).

5.2.2 RESUMPTION OF A MEETING IN IBISTRO

Some days after the meeting between Alice, Bob, and Claire, a team of developers located in Singapore browses the iBistro knowledge repository for further information regarding the application concept to be developed. While the first meeting held in Munich was rather at executive level, the two developers located in Singapore, Guying and Florian, are assigned to develop an initial prototype of the application to prepare a feasibility study for the chosen application concept. Figure 5.3 shows the timely occurrence of context and events in the Singapore meeting example.

The two developers find the decision for the advisor concept by browsing the meeting on the search key "decision". They also identify Claire and Alice as the responsible stakeholders by tracing the annotations in reverse order. Unfortunately they do not know the participants of the Munich meeting and their roles. They received their job-assignment to develop a initial prototype for the new system through a Singapore-based team lead, Tom. Tom is the head of their balanced team and responsible for issues related to the application domain and organization/ management.

Subsequently, before developing the prototype, Guying and Florian plan to have a closer look at the other option that was discussed (the computer catalog), as understanding the discarded option would likely improve the understanding

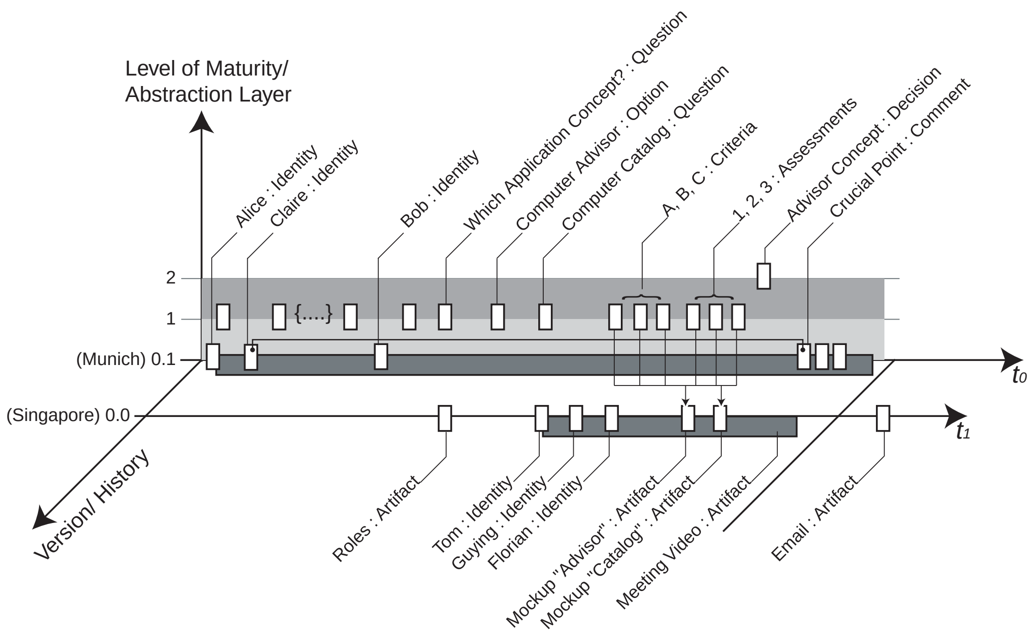


Figure 5.3: Flow of events in the 'Singapore' remote meeting example with link to the preceding meeting held in Munich. The links shown in the preceding Munich-meeting were created during the post-mortem process by the Munich meeting champion.

of the overall problem and the criteria against the options were assessed. Thus, they check the criteria against which both alternatives were assessed. To get an overview of the arguments and discussion preceding the decision for the advisor concept, the two programmers use iBistro's graphical meeting view which shows much of the meeting's content in a graph along with the relationship of events, content, artifacts, and stakeholders. In doing so, they quickly identify Bob, the third participant of the meeting. They find that Bob's comments heavily affected the decision and that most of the technical constraints and rationale for the chosen concept are actually introduced by Bob. During their brainstorming session, Florian and Guying reveal some open issues regarding the prototype. They further identify a list of both technical and domain specific questions to be clarified before starting development. To support discussion and visualize their understanding of the problem at hand, Florian and Guying draw some mockups of the planned application and store them as artifacts in iBistro.

The time difference between Singapore and Munich is seven hours and the meeting in Singapore takes place in the morning. Thus, it is difficult to get into synchronous contact with Bob. Consequently, they decide to stop the meeting, knowing that all the critical information, issues, and open questions are saved.

Florian is the *meeting champion* for the meeting just held. The meeting champion is responsible for annotating the captured video stream during the post-mortem process. Florian navigates through the meeting video using the events collected throughout the whole meeting. As the MEETINGVIEW creates events for any reference to the dated back meeting in Munich, Florian can easily identify the stakeholder for a specific issue. At the end of the post-mortem process, Florian creates messages for the individual stakeholders that point to the list of open issues in their domain. By doing so, the system now also knows about the current status and can notify subscribers.

The follow-up on the questions and issues generated during the Singapore meeting is later done by Tom, the team lead. Tom sends the open questions, annotated by references to the iBistro knowledge repository, to Bob located in Munich. Bob is now able to asynchronously browse the foregoing discussion held in Singapore to close some of the open issues.

5.2.3 A DISTRIBUTED MEETING IN IBISTRO

After the two meetings held at Singapore and Munich, the concerned stakeholders, Florian, Guying, and Bob, decide to hold another live meeting to go through the list of questions the two programmers have. They schedule another meeting to take place in the late afternoon, Singapore time, to catch Bob in his early working hours at Munich. The central issue in the scenario arises from the community distribution, which has a more drastic effect in a live meeting. Problems result from language differences (even though English is the common project language) and from the different context regarding the work environment and educational background (programming vs. banking).

iBistro supports the three project members with electronic meeting support. The electronic whiteboard lists an initial agenda, which prevents misunderstanding due to language or audio problems. The whiteboards also display any comments and ideas from both locations. Thus, the meeting uses a richer communication media compared to traditional video conferencing alone. The programmers are able to show drawings of a prototype, use cases, and other artifacts to Bob. They can be assured that any interaction is automatically captured and stored. This is important, as the two programmers might not be able to understand all aspects in the first run and, for instance due to cultural differences, might not be willing to clarify misunderstandings immediately. The captured meeting, including audio, video, snapshots, and events, is structured as meeting minutes. These minutes are also accessible from Munich. Thus, Bob is able to have a look on what the two programmers showed and can comment, clarify, or subsequently revise the meeting minutes accordingly.

The question whether to use a computer advisor or a computer catalog as an application concept implies many further questions and options which were discussed previously. The expected outcome for this meeting, however, is to provide the two programmers and specialists with all information needed to start their work on writing a technical paper for the application to build. This paper shall further discuss the two application concepts, list the pros and cons of the two concepts (including cost, development time, risk), and finally recommend a solution. Much of the information needed to set up an agenda for this meeting as well as to create an initial list of questions is already available in iBistro. By browsing iBistro using the indices created automatically (timestamps, meeting events) or manually (during post-processing), the original argumentation can be viewed as a video-stream. By using this information and working with it, the knowledge, interlinkage, and indices get increasingly rich through working with the system over and over again.

5.3 RELATED WORK

iBistro aims at solving problems in the application domain of distributed software development by offering a common infrastructure (or system architecture) to integrate HCI and CSCW into a knowledge management system.

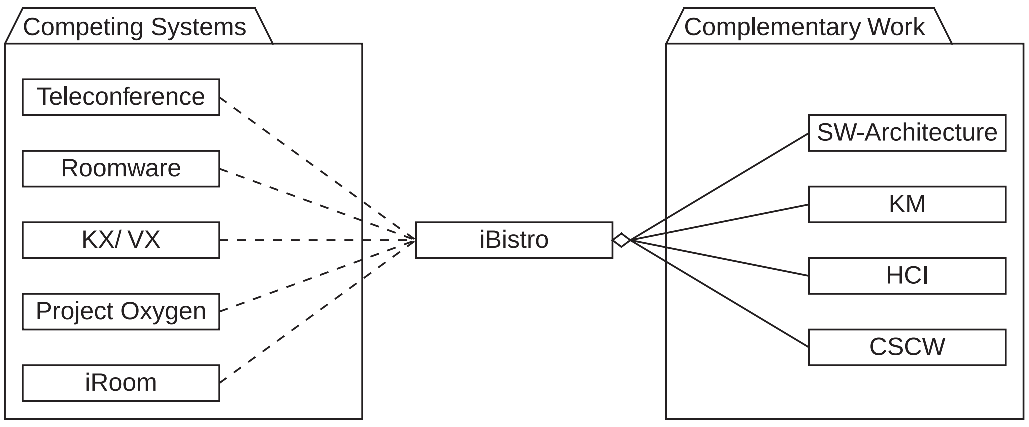


Figure 5.4: Overview of related work – competing versus complementary work.

In the following, we distinguish between self-contained and *competing research*, such as commercial groupware, CSCW systems, or tools that basically aim at solving a related problem as iBistro. These systems are potentially influential to our work but most of them are not used directly in iBistro (for instance due to availability or cost). *Complementary research*, in contrast, is likely to be used in the iBistro project. (See Figure 5.4.)

5.3.1 LIVE SINGLE-SITE MEETING SUPPORT

Specific tools for team and group support in meeting rooms are much older than CSCW and in fact much older than computers at all. Meeting room equipment, such as a blackboard (chalkboard), overhead projectors, flipcharts, or whiteboards are used in schools, at universities, or in industry for a long time to help illustrating ideas, to display topics or keywords to be visible for all participants, or in general, to support participant's retrospection. As those tools provide almost no predefined process and leaves the way they are used open to the facilitator, usage of the tools may degenerate and negatively affect the culture and purpose of meetings or lectures. For instance, especially at colleges and schools, some teachers use blackboards to write down the whole lecture word-by-word, therefore limiting the possibilities of the meeting (of type "lecture") and neglecting the original reason why people come together, which is interaction. The benefit for the participants of actually meeting in person is reduced with the reduced amount of participant interaction. However, given proper usage, blackboards, flipcharts, and whiteboards are great support for meetings. They are widely accepted and available, simple to use, and relatively cheap. Additionally, the maximum freedom in using for instance a flipchart and the absence of any pre- defined process makes them usable in almost any environment and supports creativity in using them.

Electronic support for single-site meetings can be divided into two categories: First, so called liveboards, such as the SMART Board^{TM} [SMART, 2000b], are interactive computer displays with integrated touch-sensitive tableaus. The user interacts with the SMART Board using a particular pen or her fingertip instead of a keyboard and a mouse. Liveboards can display any desired information, as they are simply computer displays. For instance, a liveboard can be used to pop-up an UML diagram by opening a UML modeling tool, such as Rational Rose^{TM}. The degree of meeting support depends on the software provided.

The second category are whiteboard-based capturing systems. Examples include ZombieBoard [Saund, 1998; Moran et al., 1999] or Hawkeye^{TM} [SMART, 2000a], a commercial whiteboard capture system based on ZombieBoard. Those systems simply capture the content written on a whiteboard by digitalizing it. The resulting image then is accessible as a bitmap.

Liveboards, as opposed to "zombie"-boards, enable displaying related digital material and artifacts. They also allow for the use of software to support an ongoing meeting. Both liveboards and zombie-boards allow a natural way of interacting with the system and are used more like conventional roomware than like a computer.

5.3.2 LIVE REMOTE CONFERENCING

Live remote conferencing, also referred to as video or teleconferencing, experienced remarkable gains in demand especially after the events of September 11th 2001 [Kaufman et al., 2002]. Larger companies, for instance Accenture, tried to reduce travel whenever possible. Video conferencing promises to transfer the advantages of human face-to-face communication to an electronically supported, distributed medium. The demand for face-to-face meetings is derived from the insight that human communication is more than just words. By looking at their conversational partner, people gather much richer information, such as gesture and facial expression. This sort of "contextual information" [Moran and Dourish, 2001; Winograd, 2001] provides estimations about intentions, whether the opponent in a discussion tends to agree or rather disagrees, or if the speaker is clear or not.

The Portland Experiment [Bly et al., 1993] was one of the first video-based systems to support informal ad-hoc meetings. Between 1985 and 1988, two Xerox

PARC laboratories were distributed between Palo Alto in California and Portland, Oregon. Research groups were distributed between both locations to investigate collaboration over distance using video-conferencing. The research focus was on the technical infrastructure, design methodologies for collaboration systems, and on the mediaspace. The insights from this research are exemplary for video-conferencing systems in general:

- the unification of two physically divided and distributed spaces to build one common virtual space experienced resistance from users;
- using technology mainly to simulate face-to-face meetings has been reported as the causal reason for user resistance;
- a shared electronic drawing board which was used in both locations, however, was not affected negatively;
- the issue of common resource usage versus reduced dependency of the distributed locations in terms of the technical infrastructure was a neglected issue. This resulted in developing the so called "Object Server";
- from the organizational and managerial point of view, there was no distributed decision making. As the organizational hierarchy was shifted towards Palo Alto, Portland was not really independent in terms of deciding. This fact, however, could not be solved by the video-communication, which was not able to lower the necessity of physical presence in Palo Alto to participate in decision making;
- video cameras in private areas created a floating transition from public to private spaces. The so called "ethic video" was accepted by the users, as the hook-up of remote users was ever apparent and thus transparent for local users;
- the experiment delivered good results with control by management and in decentral control, as part of the control always lied in the hands of the employees and the assessment of remote employees has been reported being less subjective.

A similar experiment at Accenture between Northbrook (near Chicago), Illinois, and Palo Alto reported comparable results (Magic Wall [Hughes, 1999]). However, due to improved bandwidth, Magic Wall is often used for distributed conferencing and team meetings. In both locations, Northbrook, Il., and Palo Alto, Ca., a specific meeting room is equipped with a large flat-screen computer display on the wall (the "magic" wall). The Magic Wall creates a video-supported view into the other room in the remote location. This creates a larger virtual meeting room with the remote colleagues visible face-to-face. Specific audio support is used to allow for real-time conversation between various meeting participants. Magic Wall is more than a video conferencing tool. Its computer display can be used to overlay any kind of information that is available electronically, such as sketches, programming language source code, screen mockups, or software user interfaces in general.

Today, there is a large variety of commercial and academic teleconferencing systems available. We will subdivide teleconferencing systems into commercial and academic systems. The main difference is that commercial systems tend to rely on proven concepts and technology. Academic systems, in contrast, prototypically inspect new concepts and techniques, while putting up with several problems in usability, user acceptance, and steadiness.

Commercial Video Conferencing Systems

Commercial video conferencing systems are offered at a broad range of functionality and cost, starting with small software packages or add-ons to operating systems to provide a basic functionality, such as Microsoft's NetMeeting^{TM}. Cohesive systems are available from PictureTel, PolyCom, CUSeeMe, Sony, V-Tel, and many other corporations. Many commercial video conferencing systems are similar in their basic functionality. As an illustrative example, we focus on PictureTel's [PictureTel Corporation, 2001] system. PictureTel offers two types of systems. The personal system product line enables peer-to-peer live conferencing, while the group systems promise natural meeting experience for distributed groups of people.

PictureTel's 550 personal system is based on Microsoft NetMeeting. Thus, its use is restricted to Microsoft operating systems. NetMeeting provides a picture-in-picture view with a large window to display the remote person and a small window for the local person. PictureTel allows users to share artifacts electronically, such as documents or files. As a communication infrastructure, both the Internet Protocol (IP) as well as ISDN are supported.

While PictureTel's personal system is meant to connect two conferencing people peer- to-peer, group meetings are supported in PictureTel's group video conferencing systems, called iPower 600 and 900. The iPower systems offer controlled pan-tilt-zoom cameras and microphone arrays. The microphone arrays identify the position of the voice's source and thus enable speaker tracking with the video camera and improved speech quality. iPower systems also feature TV-based interfaces to connect larger screens. Further, PictureTel's so called "Enhanced Continuous Presence" technology allows users to display multiple meeting sites on-screen at the same time. Remote users can choose six different screen layouts, like full-screen, 2-way (side-by-side, above/ below), 4-way quad, 1+5 (1 large window, 5 smaller windows), and 9-way. The system also features meeting recording and on-demand viewing.

Research Systems for Video Conferencing

Academic video conferencing systems explore many aspects of video conferencing that are more difficult to grasp and go beyond basic support for video and audio transfer over a network. Research systems attempt to support sophisticated features at the risk of behavior which might bother people working with the system. Buxton, Sellen, and Sheasby in their book "Video Mediated Communications" embrace research goals of academic video conferencing tools, such as Hydra, LiveWire, Portholes, and BradyBunch [Buxton et al., 1997]:

- establishing eye contact;
- awareness of others;
- who is attending to whom;
- parallel conversations;
- ability to hold side conversations;
- perception of group as a whole;
- ability to deal with shared documents and artifacts.

It is obvious that these goals are difficult to achieve and attract some resistance from users. Prototypes also raise new issues, for instance privacy (which collides with awareness of others), technical challenges (e.g., establishing eye contact or video and audio quality), and organizational and social implications (e.g., perception of group as a whole). Monitoring eye contact and trying to identify "who is attending whom", for instance, lead to frequent camera switching and flickering and zapping video presentation. (More information on user resistance is listed in [Sellen, 1995; Buxton et al., 1997].

While most of the results achieved are far from reaching maturity for a commercial product, they address key issues in making video conferencing systems usable for a broader community of users.

5.3.3 ASYNCHRONOUS GROUPWARE SYSTEMS

Argumentation, negotiation, and decision support systems enable asynchronous collaboration. All those systems are somehow based on the ideas of bulletin-board systems (b-boards) or the Usenet. In a few words, b-boards allow various users to ask questions and to reply to questions. Any user may browse the set of questions and answers at anytime and thus answer questions or read asynchronously. A similar system can also be used to store information in a rationale-based approach. This approach has been followed in IBIS [Rittel and Weber, 1973] and gIBIS [Conklin and Begeman, 1988], using issue-based argumentation, and by [MacLean et al., 1996], using the Question, Option, and Criteria (QOC) approach.

ReQuest [Dutoit and Paech, 2001a] is a tool support for requirements engineering through a process called *rationale use-case specification*. ReQuest is a web-based collaborative application to write use-cases and model questions, issues, challenges, and justifications using question, option, and criteria (QOC) tables (as opposed to QOC diagrams). ReQuest focuses on supporting requirements specification and the rationale behind them. Beyond providing tool support, ReQuest offers its own method of requirements specification by combining use cases and rationale information. Moreover, ReQuest guides participating stakeholders by modeling the process in the tool. This also integrates used principles such as use cases and QOC. ReQuest is also integrated with discussion support (like a b-board) to enable and track discussions related to requirements engineering.

5.3.4 RELATED PROJECTS

Carnegie Mellon University's BARN

The BARN meeting room [Anthes, 2003] is an upcoming prototype conference room capable of recording everything that happens during a meeting through an array of microphones, cameras, projectors, and other equipment. The Barn was specifically designed to support small local teams in elaborational design meetings. Barn supports brainstorming, idea generation, knowledge generation, and knowledge transfer. Conference participants register their presence by radio-frequency identification tags, while wearable sensors allow the Barn to confirm their identity and constantly track their location. The adjustment of the rooms' environmental settings, so far lighting and microphones are calibrated accordingly to attendees' physical position, is called *social geometry* in BARN. A key component of the meeting area is a digital whiteboard outfitted with an interactive display, the *Thinking Surface*. The Thinking Surface is used to display concepts. Major decisions or brainstorms are flagged in meeting logs when someone pushes a TWI button ("that was important") on his computer. TWI markers are useful for people who miss meetings and need to be brought up to speed quickly.

The BARN is a very recent research effort. So far[1], there are no publications available to summarize the actual features of the BARN. However, there is a list of missing features and upcoming extensions for BARN. For instance, future BARN

[1]As of 9th March 2004.

research will focus on avoiding contradictory decisions among semi-independent subgroups within large project teams.

Stanford's iRoom

The iRoom project developed at Stanford University [Fox et al., 2000] presents an approach to integrate information appliances into an interactive workspace called the *iRoom*. The research project offers a project infrastructure and software architecture built around three SMART Boards. The iRoom smoothly integrates the wall-mounted displays with laptops and PDA's brought into the room.

The iRoom software infrastructure consists of three basic components: the EventHeap, MultiBrowsing, and PointRight. The EventHeap provides the basic communication and coordination mechanism for the room. MultiBrowsing manages content across multiple displays using the EventHeap. PointRight manages user interaction by integrating various input devices, such as mice, keyboards, touch panels, and other input devices in the room. The three components together enable users to interact with information that is typically handled in workstation environments. Figure 5.5 illustrates the CIFE iRoom configuration and architecture.

MIT's Project Oxygen

Project Oxygen is a joint project of the MIT Artificial Intelligence Laboratory (MIT-AI) and the MIT Laboratory for Computer Science (MIT-LCS). Oxygen aims at yielding a profound leap in human productivity by enabling people to use spoken and visual cues to automate routine tasks, access knowledge, and collaborate with other people "anywhere, anytime". Hence, Oxygen uses ideas from ubiquitous computing [Weiser, 1991] and pervasive computing to achieve its goals. Oxygen, however, is more a technology-driven approach with a strong focus of integration of various new technologies, sensors, and virtual and augmented reality. Research and prototypes of Oxygen mainly focus on what would be called the input-side in iBistro. Oxygen offers a variety of human-computer interaction interfaces, such as speech recognition and hand-held devices and aims at integrating ubiquitous computing, wearable and mobile computing, and augmented reality efforts.

Discussion

The three related approaches focus on development and appliance of sensors and gadgets to be used in an augmented meeting space. They choose the support of synchronous (live) meetings of a small team in non-distributed settings as application domain. In contrast, this dissertation focuses on the integration of arbitrary sensors and gadget, including third party developments potentially from

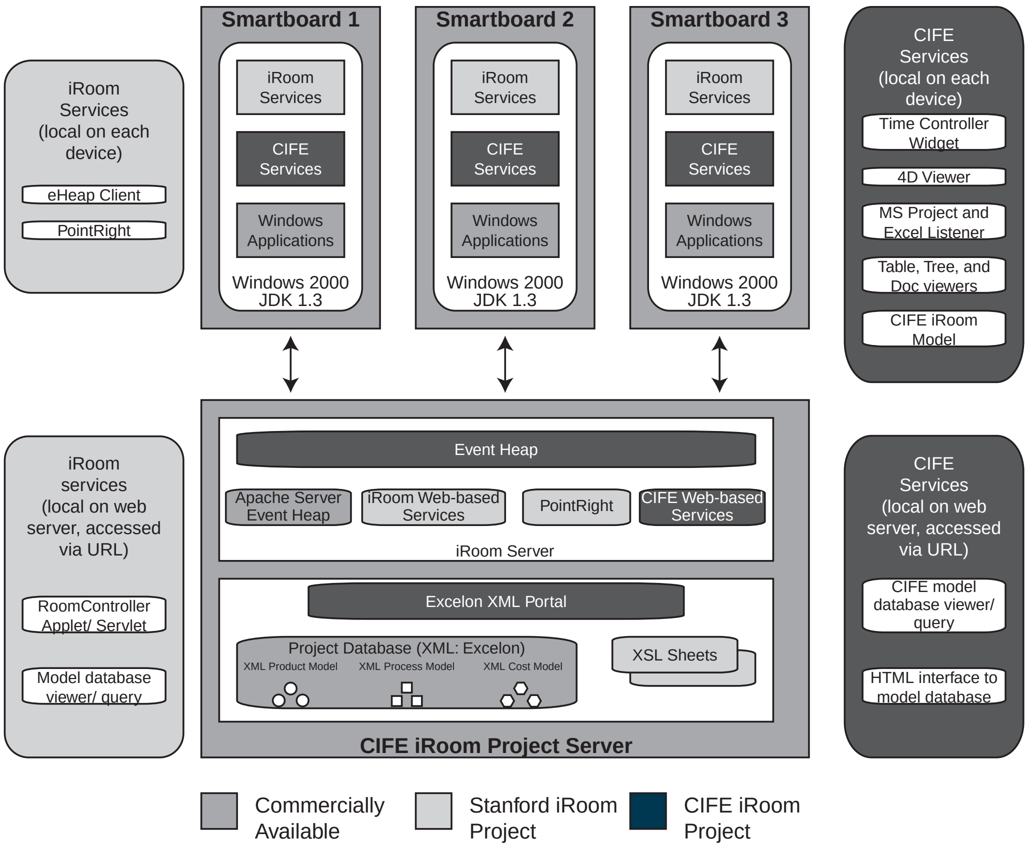

Smartboard 1
Smartboard 2
Smartboard 3
iRoom Services
CIFE Services
Windows Applications
Windows 2000 JDK 1.3
iRoom Services (local on each device)
eHeap Client
PointRight
CIFE Services (local on each device)
Time Controller Widget
4D Viewer
MS Project and Excel Listener
Table, Tree, and Doc viewers
CIFE iRoom Model
iRoom services (local on web server, accessed via URL)
RoomController Applet/ Servlet
Model database viewer/ query
Event Heap
Apache Server Event Heap
iRoom Web-based Services
PointRight
CIFE Web-based Services
iRoom Server
Excelon XML Portal
Project Database (XML: Excelon)
XML Product Model
XML Process Model
XML Cost Model
XSL Sheets
CIFE iRoom Project Server
CIFE Services (local on web server, accessed via URL)
CIFE model database viewer/ query
HTML interface to model database
Commercially Available
Stanford iRoom Project
CIFE iRoom Project

Figure 5.5: Overview of the CIFE iRoom configuration and architecture [Fischer et al., 2002]

the three cited projects. We offer an standardized way of integrating data, information, and knowledge into a common distributed knowledge space (repository) for several distributed teams. To achieve the goal of supporting several globally distributed teams, this dissertation offers a common distributed software architecture and meeting post-mortem structuring, while integrating mainly third-party context-aware devices and sensors.

All three competing systems lack a unified way of integrating various knowledge sources to capture information from synchronous or asynchronous communication and retrieve that information in related meetings in different locations. Further, there is no support for transforming low-level raw information, such as a meeting video and audio stream, into high-level knowledge, such as machine readable meeting minutes. Live systems run the risk of being restricted to simple communication, just like telephones, while argumentation systems miss much information, as they are cumbersome to use, especially during a meeting. The discussed systems also do not offer an extendable knowledge structure and taxonomy to capture and organize knowledge and information and deal with community issues that occur in global teamwork.

The three systems support local meetings respectively brainstorming sessions with a strong focus of integration of HCI and context-aware devices. So far, issues resulting from the distribution of teams or from scale are not addressed.

Table 5.1 summarizes related research projects:

Requirement or Attribute	BARN Project	Project Oxygen	CIFE iRoom	iBistro
Synchronous collaboration	yes	yes	yes	yes
Asynchronous collaboration	no	no	no	yes
Team support	yes	no	no	yes
Distributed team support	no	no	no	yes
Large projects/ several teams	no (upcoming)	no	no	yes
Meeting support	yes	no	yes	yes
Roomware support	yes	yes	yes	yes
Context-aware system	yes	yes	no	yes
Augmented reality system	yes	yes	no	no
Knowledge space/ base	no	no	yes	yes

Table 5.1: Comparison of related projects.

CHAPTER 6

IBISTRO ARCHITECTURE AND FRAMEWORK

"AI has been thought controversial because it challenged the uniqueness of human thought, as Darwin challenged the uniqueness of human origins.
The boundaries of AI continue to expand rapidly, settling the controversy for those who know the evidence."

[HERBERT A. SIMON]

6.1 CHAPTER OVERVIEW

To fulfill the functionality of iBistro as described in the previous chapters and in the scenarios, we extend the standard blackboard model to enable concurrency and distribution.

The *distributed concurrent blackboard architecture* (DCBA) [Braun et al., 2003] shown in the figure is built up from two local blackboard installations at the client site ("client-site teams") and at the remote location ("offshore teams") in the example shown in Figure 6.1[1]. The distributed concurrent blackboard provides a transparent means for knowledge sources and users to access one single 'portal', regardless of its actual technical implementation. A local client-site team works at the client's place of work, while an offshore software development team works remotely in a different distant location.

Figure 6.1 shows an informal overview of the iBistro system.

[1]The exemplary scenario of two iBistro sites, one local at the client site and a remote development site, is used as a graphic example during this chapter, regardless of the fact that an actual installation might similarly consist of more than two sites.

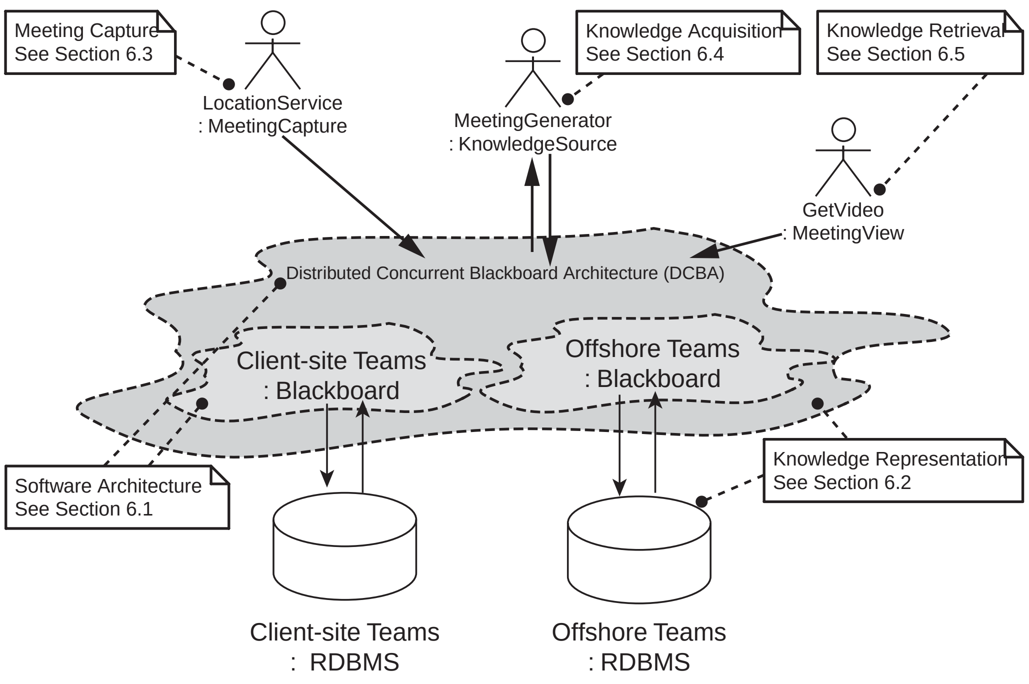


Figure 6.1: Informal overview of the iBistro system and this chapter. (A more detailed view of the overall architecture is displayed in Figure 6.12 in Section 6.7.) The figure shows the *distributed concurrent blackboard architecture* and concepts for knowledge representation and storage, capture, acquisition, and retrieval.

The figure displays five fundamental concepts for iBistro which are detailed as described in the following overview:

1. The Distributed Concurrent Blackboard Architecture

The *distributed concurrent blackboard architecture* (DCBA) serves as a primary data repository. *Local components* (for instance the MEETINGGENERATOR or LocationService in Figure 6.1) are running on machines hosted by the local project teams. They are connected to the blackboard and local events are captured and stored in the blackboard as they occur. The blackboard is *distributed* because knowledge sources can connect to one distributed transparent repository. The DCBA itself is built from several servers which together form the overall system as a primary and transparent communication and storage infrastructure. The blackboard is *concurrent* because several components can access the repository at the same time, and the local blackboards can operate independently. The blackboard builds up a *knowledge space* for data and knowledge storage.

2. Knowledge Representation and Storage

Data and knowledge stored in the knowledge space is recorded accordingly to a taxonomy of data items. The hierarchy used is represented in the layers (or levels) of the blackboard model. Basic types of information (video streams, context events) are stored at lower levels of the blackboard, while higher-level information (hypothesis, solutions) is stored at higher-levels. Such partitions are necessary to structure the distributed domain knowledge, which is represented in the collection of knowledge sources. The structure of the levels is also necessary to control the data on the blackboard and to organize the levels of hypotheses.

3. Meeting Capture, Capture Components

Meeting and information *capture components* capture a particular type of contextual information, for instance sensor-based events. This information is then offered to the system, regardless of its potential use. In iBistro, specific capture components capture various types of context, such as people entering or leaving the meeting room (*location-based*), people using specific equipment in the electronic meeting room, such as the electronic whiteboard (*activity-based*), or access to project-relevant artifacts, such as source-code (*artifact-based*), and many others. All capture components have in common that they track information that can be easily electronically recorded and caught. The resulting information is stored as a basic type of "knowledge" at a low and raw level of abstraction (data items). The video capture component, for instance, simply records audio and video of a meeting and puts the resulting video-stream (as an artifact) into the knowledge space.

4. Knowledge Acquisition

Knowledge sources pick up basic or abstract data items (like a whiteboard snapshot stored in a bitmap) and work on them, potentially by using and combining the information captured by several different meeting capture components. Knowledge sources exclusively work with information stored on the knowledge space. They are only able to communicate indirectly with other knowledge sources using the blackboard.

The MEETINGGENERATOR is a concrete knowledge source that works on the diverse information created during one single meeting, including the recorded video stream.

5. Knowledge Retrieval

Knowledge views provide access to the contents and structure of the knowledge space. Similar to the model-view-controller paradigm [Burbeck, 1987], a variety of knowledge views provide different visualizations of the state of the knowledge space. In iBistro, the knowledge space can be viewed by stakeholders and responsibilities, or by events, or by replaying a meeting video-stream. Knowledge views are used in particular to provide a human-computer-interface (HCI) to the information and knowledge stored in a knowledge space.

6.2 THE DISTRIBUTED CONCURRENT BLACKBOARD ARCHITECTURE (DCBA)

The *distributed concurrent blackboard architecture* (DCBA) is an extension of the standard blackboard model. Two major design limitations of the standard blackboard model necessitate its extension for iBistro: first, the standard model does not allow concurrent work on the blackboard. Second, the standard model does not allow distribution, especially a distributed knowledge space (blackboard). (See Section 2.5.)

The specific characteristics of both (informal) meetings and software development substantiate our design decision for the DCBA. The scenarios in Chapter 5 show that there is no single or consistent way, or algorithm, to capture the richness of an informal meeting. In the informal meeting, numerous and heterogeneous types of 'events' occur concurrently. Moreover, information such as stakeholder names, task assignments, or responsibilities for e.g. a specific source code is related to organizational or project knowledge, building up a complex network that spans across multiple stakeholders, meetings, teams, problem domains, locations, and potentially organizations and projects. The DCBA provides a means to gather the information and store and process the mesh of knowledge. We sacrifice the claim of unconditional consistency for the benefit of completeness and especially information richness. Hence, our goal in iBistro is to capture as much of a meeting as possible, while only the part of the information which can be understood and used is reused to further build solutions. By adding new and improved components to work on information not used so far, the system improves step-by-step and is able to better capture, store, and represent the diverse events in human communication.

Software development is a problem-solving activity. In a large software project, many different distributed stakeholders contribute to the resolution with their individual knowledge of how to find a (partial) solution to (parts of) the problem. During the process of finding a resolution or partial solutions, stakeholders gather and contribute many different types of information. For instance, one single source file that builds a partial solution for the whole system is built using many different types of contributions, such as programming expertise, application domain knowledge, social skills, and many others. The final version of the artifact (the executables and companion files) eventually contains many, but not all, of the contributions made. These contributions, however, are often not used as contributed initially, but in some revised and improved version. Further, the flow of events and contributions made is not predictable. Thus, finding a resolution is an *opportunistic*, as opposed to *systematic*, process.

In other words, the process of building software is reminiscent of the process of a puzzle assembly from little pieces "step-by-step". It can be seen as *knowledge* *assembly*, in contrast to the (non-opportunistic) *search for solutions*.

This distributed view of the problem domain suggests an approach based on the *blackboard model* [Buschmann et al., 1996]. The blackboard model supports opportunistic problem solving by incorporating a variety of expertise from complementary domains, embedded in *knowledge sources*. Just like on a software development project, a central component directs knowledge acquisition and problem solving by scheduling the "experts". Hence, the blackboard assembles solutions that satisfy a given threshold; there is no exhaustive computation until the very best solution is achieved. The blackboard will likely come up with several solutions solving the problem. (See Section 2.5 for more information on problem solving in the standard blackboard model.) The distributed concurrent blackboard architecture is based on this architectural model. It consists of several specialized subsystems which assemble their knowledge to build a possibly partial or approximate solution.

The blackboard pattern as described by Buschmann et al. consists of three fundamental classes: the `Blackboard`, `Control`, and the `Knowledge Sources`. In iBistro, the standard blackboard pattern has been extended by five fundamental classes: `Strategy`, `DatabaseConnector`, `Capture`, `History`, and `View`. In iBistro, the blackboard class is further split into a `LocalBlackboard`, which represents the suitable repository for local balanced teams and extended by a `DCBAConnector` to interface remote sites and their local blackboard implementations.

Figure 6.2 illustrates the DCBA and its components as an extension to the standard blackboard model. In the following we give detailed description of the classes used in the DCBA architecture.

6.2.1 THE *BLACKBOARD* CLASS

The blackboard is a common data structure in the standard blackboard model. One single blackboard serves as a medium for all communication within the system. The blackboard can store any kind of information derived from its basic type (`DataItem`). The DCBA provides a common interface and communication infrastructure for connected components and users (`DCBAConnector` in Figure 6.2). The DCBA hence is a virtual connector made from many blackboards, for instance "Client-site Teams" and "Offshore Teams" in Figure 6.1 (Page 100). While the `DCBAConnector` provides a transparent way of accessing the system and hides the underlying layout, the local blackboards perform actual data storage. The blackboard is organized into several layers in which data is organized. The layers (and therefore the blackboard) are a non-permanent data storage, similar to a caching device. The objects cached in the blackboard are stored persistently in a relational database (the `DatabaseConnector`). When a local blackboard system is not available, crashes, or is shut down, the contents of the black-

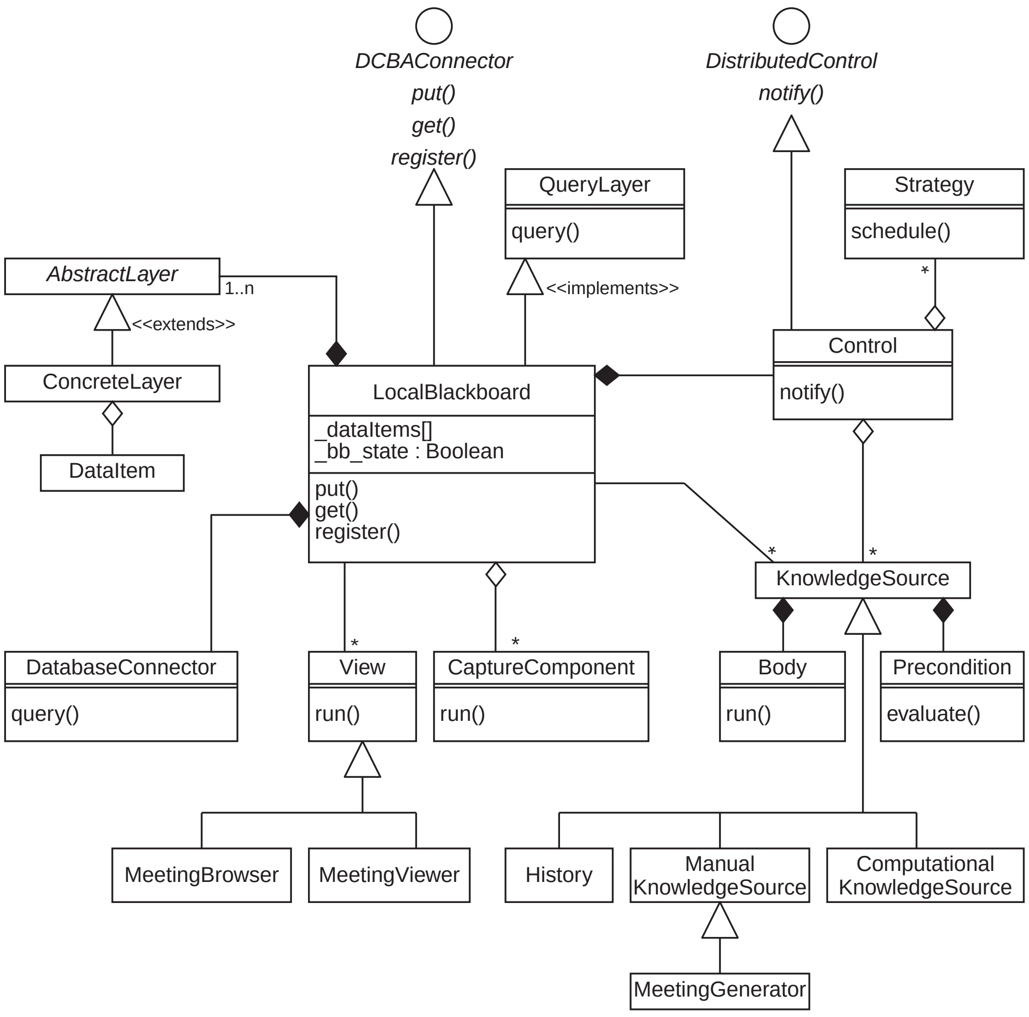


Figure 6.2: Classes in the *Distributed Concurrent Blackboard Architecture.*

board are still stored in the data base. When the local blackboard is started again, the layers are empty at first. By working with the blackboard, the used objects are retrieved from the SQL data base and cached in the layers of the blackboard accordingly to their level of abstraction. An important side effect is, that *query objects* can be created in the structured query language (SQL). Hence, all our blackboard implementations understand SQL as a query language (`QueryLayer`).

6.2.2 THE *DCBACONNECTOR* CLASS

The `DCBAConnector` provides the interface layer for all components to access the DCBA architecture. It provides a central way of accessing the DCBA and directs the operations to the local instance of a blackboard. The DCBA also directs the replication of knowledge from a local instance of a blackboard to another. Queries are directed to a local instance of a `DCBAConnector`. The Query is forwarded to the local blackboard as well as to other blackboards known to the `DCBAConnector` layer. The responses then are replicated to the local instance to which the requesting knowledge source is physically bound. Thus, remote knowledge items are replicated on first use (*lazy copy*).

6.2.3 THE *CONTROL* AND *STRATEGY* CLASSES

The control and strategy classes direct the process of knowledge building by controlling and directing knowledge acquisition. As described in Chapter 5.3 and especially in Figure 2.6, the Control class evaluates which knowledge sources are to be invoked, given the current state of the blackboard. The current state of the blackboard or the relevant portion of the blackboard that is meaningful for knowledge source execution is also called *context*. The context is an important part of the focus of attention.

The Focus of Attention

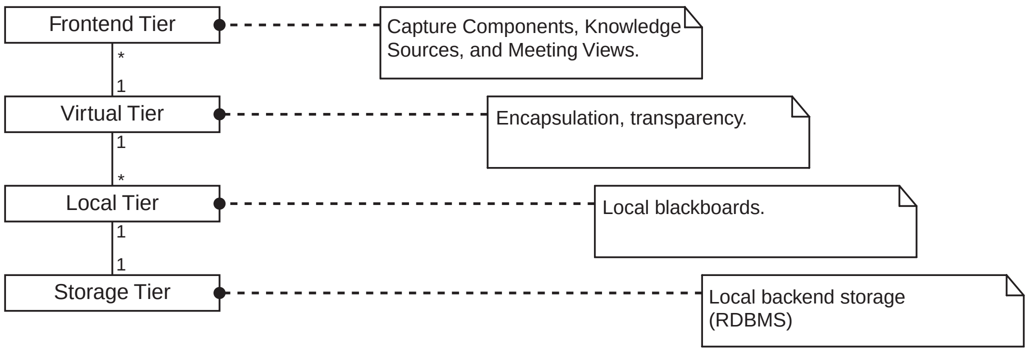
Knowledge source activation and context selection is conducted following the so called *focus of attention* principle [Nii, 1986a; Nii, 1986b]. In iBistro, various kinds of global information are kept in the DCBA, including the control information used by the control components to identify the focus of attention. The focus of attention indicates the next objects or components, or combination of both, to schedule for execution. In other words, the objects on the blackboard are the *context*, the knowledge sources represent the *activity* (or action). The activity, hence, happens *within the given context*. If σ is the function that creates the focus of attention, then $\sigma(\{O_1, \ldots, O_n\}, \{c_1, \ldots, c_n\})$ is the focus of attention σ, with O_n being the bound objects and c_n the involved components. The result of the function σ is a set of objects O_n and knowledge source components c_n.

The selection of the focus of attention also influences which knowledge source is activated next. Therefore, the choice of a focus of attention affects the sequence of knowledge source activation. The selection of a focus of attention is triggered by external events, especially `put()` operations on the blackboard. The actual selection of a focus of attention is directed by the `Strategy` component in the DCBA. The `Control` component uses the focus of attention to schedule the next activity.

The solution in legacy blackboard systems is built one step at a time. In the DCBA, many knowledge sources concurrently work on objects on the blackboard(s), potentially overlapping. This is made possible by strict version control. (See Section 6.3.2.)

The Data and Control Flow in the DCBA

In the DCBA, the distributed control class acts as a broker. The distributed control (`DistributedControl`) delegates component registration to the local implementations (`Control`). Figure 6.4 depicts the flow of events and collaboration of components in the DCBA in a collaboration diagram.



Frontend Tier
*
1
Virtual Tier
1
*
Local Tier
1
1
Storage Tier
Capture Components, Knowledge Sources, and Meeting Views.
Encapsulation, transparency.
Local blackboards.
Local backend storage (RDBMS)

Figure 6.3: The 4-tiers in the iBistro Architecture.

The architecture can be seen as four tiers. (See Figure 6.3.) On the *frontend tier*, users and components, such as knowledge sources reside. The frontend connects to the *virtual tier*. The virtual tier encapsulates the underlying concrete layout and composition of the system. Thus, frontend components do not need to know about the actual number of blackboards the DCBA is made of, or about their location[2]. The *local tier* is composed of the dynamic collection of interconnected local blackboard infrastructures. A local blackboard infrastructure consists of the local blackboard, control, and strategy components. The composition of this tier is dynamic because local blackboards might be stopped or unavailable, while others may connect to the system dynamically. The fourth tier, *storage tier*, is responsible for persistent data storage in a relational database.

The sequence shown in Figure 6.4 illustrates the flow of events in knowledge source scheduling and focus of attention in the DCBA.

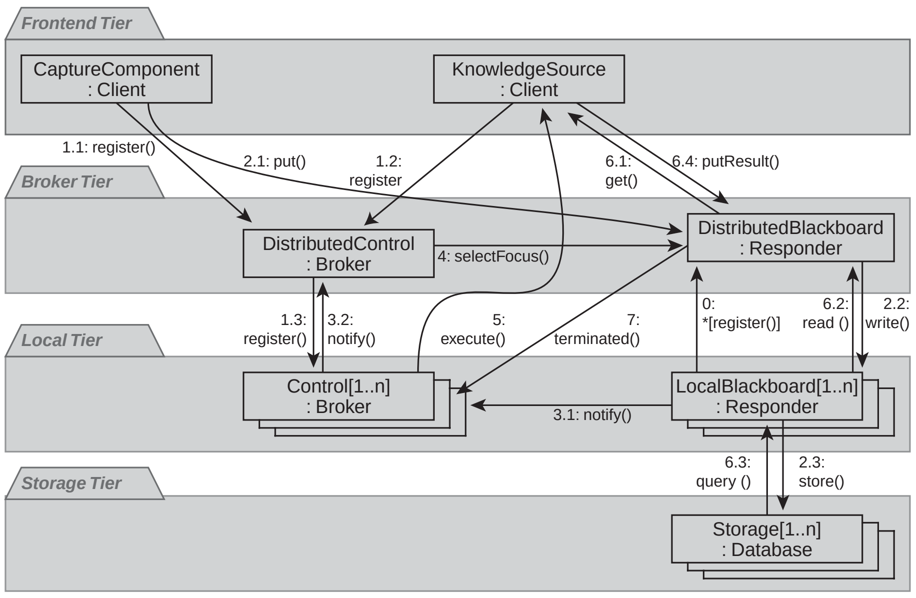
6.2.4 THE *HISTORY* CLASS

The `History` class logs any activity on the local blackboard, as well as the scheduling actions taken by the Control class. As iBistro is chiefly meant to be an experimental system for global software development, the historical entries are used for analysis and improvement of the system. Moreover, the entries can be used by specific strategy classes to optimize the scheduling strategy for knowledge

[2]As knowledge sources typically run in a physical environment provided by the IT services of a location (i.e., client-site or offshore), the components may connect to any local DCBA connector to connect to the virtual tier.

sources. This is specifically used to gather performance related information for data replication between the different local blackboard implementations.

Events captured in iBistro can cause subsequent events, as events might be used to trigger an activity. This creates *networks of events*. One knowledge source, for instance, might create an event if one specific user of the system enters the meeting room. Another knowledge source then possibly notifies two other users that this person is available for a chat. While events may span local blackboards, the history entries will be recorded locally only. The history entries itself, however, are stored in the DCBA, and hence are available globally (for instance for the `DistributedControl` class). The history of such context events, however, is considered an important additional source of information. By the history of context we understand how the events develop and span over time; which events likely create other events or trigger user activity, and so on.



0	`register()`	Registration of local blackboards at the start of the system
1.1-1.3	`register()`	Each knowledge source and capture component initially indicates the contribution it can make to the problem solving process by registering itself at a control component. While a capture component simply adds data of a specific type to the repository, a knowledge source expresses its contribution by the expected type of input, the resulting type of output, and the *precondition*. The precondition expresses a rule that delivers *true* if the knowledge source shall be applied on the given input.
2.1-2.3	`put(), write(), store()`	One or more objects in the blackboard space are added or changed. This can be done either by knowledge sources or capture components.
3.1-3.2	`notify()`	Notify control components on object change.
4.	`selectFocus()`	The local control component selects (from the information given in points 1 and 2) a focus of attention.
5.	`execute()`	Depending on the information contained in the focus of attention, the control component prepares knowledge source scheduling as follows: 1. **IF** the focus of attention is a knowledge source, **THEN** a blackboard object or a set of objects is forwarded to that knowledge source upon invocation (knowledge-scheduling approach). 2. **IF** the focus of attention is a blackboard entry, **THEN** a suitable knowledge source is chosen by checking the layer and preconditions (event-scheduling approach). 3. **IF** the focus of attention is a knowledge source **AND** an object, **THEN** the knowledge source is executed with the packed object(s).
6.1-6.4	`run: get(), read(), query(), putResult()`	The knowledge source is executed without being controlled by the `Control` component. During execution, the knowledge source might use the blackboard or other sources of information (such as local filesystem, the web, or databases) to gather information other than delivered along with the focus of attention, or put data on the blackboard.
7.	`terminate()`	On termination, the knowledge source puts its work product on the (local) blackboard. The control component is notified on knowledge source termination.

Figure 6.4: The flow of events and control in the DCBA (collaboration diagram).

6.3 KNOWLEDGE STORAGE & REPRESENTATION

The collection of local blackboards, consolidated in the DCBA, serves as a repository for all information. While capture components only add data to the blackboards, knowledge sources produce changes on the blackboards based on the data already available. Interaction among the knowledge sources is only possible through the blackboard. The blackboard consists of objects from the solution space and information used by the system itself, such as control data.

The objects on the blackboard are hierarchically organized into *levels of analysis* [Nii, 1986a]. The hierarchical information on objects is used by the knowledge source's precondition to identify suitable input data. This information, called *properties* is contained in the *knowledge taxonomy* used in iBistro. The levels of analysis divide the storage space into several *layers of abstraction*: the knowledge space is partitioned into multiple hierarchies.

Beyond the hierarchical order, there are two more types of relationships between objects: *named links* describe how an object is related to another object, independently of its hierarchical position; *version links* denote how objects evolve through several versions.

Eventually, the blackboard is a non-persistent storage system. It "caches" knowledge objects during its running time. To permanently store knowledge objects represented in the blackboard data structure, a relational database is the endmost tier of each local blackboard.

In the following, we describe the concepts of knowledge storage and representation: the *hierarchy and knowledge taxonomy, named links and version control, the layers of abstraction*, and finally, *persistent data storage*.

6.3.1 ONTOLOGY & KNOWLEDGE MODELING IN UML

> **Definition**[3]**.** An ontology is a formal and explicit specification of a shared conceptualization. A conceptualization refers to an abstract model of some phenomenon in the world which identifies the relevant concepts of that phenomenon. *Explicit* means that the type of concepts used and the constraints on their use are explicitly defined. *Formal* refers to the fact that the ontology should be machine understandable, i.e. the machine should be able to interpret the semantics of the information provided. *Shared* reflects the notion that an ontology captures consensual knowledge, that is, it is not restricted to some individual, but accepted by a group.

An ontology is an explicit notation of a conceptualization [Gruber, 1992].

[3]Based on related definitions in [Broekstra et al., 2002; Gruber, 1992; Gruber, 1993].

The ontology represents the potential of information and knowledge that could be represented in the system: for AI systems in general, what "exists" is that which can be represented. An ontology supports modeling of concepts in a domain and the expression of information and knowledge in terms of those concepts.

One problem in the DCBA – similar to problems in AI and expert systems – is that the ontology is never complete and will likely be changed and revised while the system is used. To approach this problem along with the problem of finding usable knowledge representations for an object-oriented system, we use the Unified Modeling Language (UML, [Rumbaugh et al., 1999]) as a semi-formal notation for ontology specification. Figure 6.5 illustrates the design of an ontology in UML and further use in the DCBA. The UML is used increasingly in the domain of problem modeling [Arlow et al., 1999; Cranefield and Purvis, 1999; OMG and Meta Group, 2000]. The UML in particular provides a graphical notation used for ontology design in the DCBA. The UML further provides a powerful and highly expressive constraint language, the *object constraint language* (OCL). Finally, several commercial and non-commercial UML design tools are available. The UML design can be either transformed directly to programming language representation, such as Java, or to the *XML Metadata Interchange* format (XMI). The *Resource Description Framework* (RDF) is originally built for the Web. The RDF is a standardized metadata (information about information) description to describe which information is actually there and to name and identify chunks of data and information. XSLT, the *eXtended Stylesheet Transformation* language, enables and empowers interoperability by offering a standardized way of describing and programming XML transformations to other data formats. An XSLT stylesheet, for instance, would allow to transform an XML file (data) to HTML, while another XSLT stylesheet would transform the same data to the Portable Document Format (PDF). Both, XSLT and RDF are standardized by the W3C.

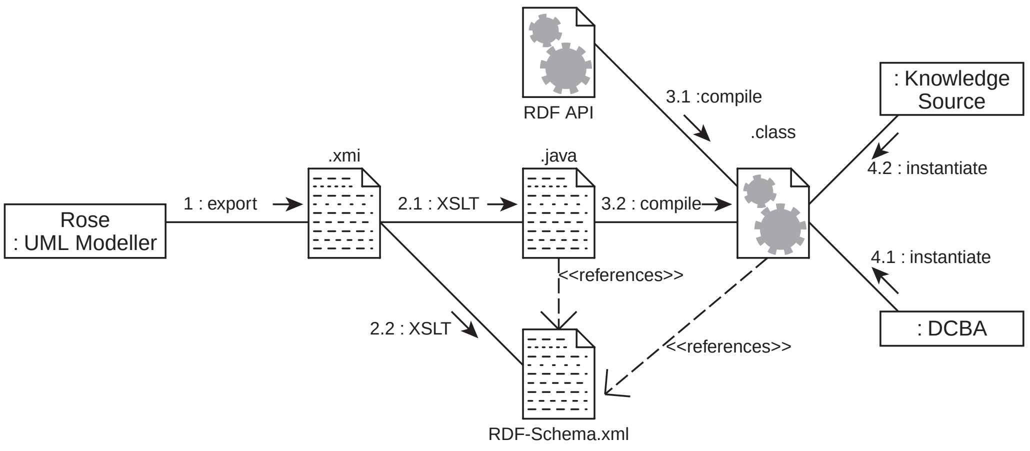
The process shown in Figure 6.5 illustrates the design of an ontology in UML and further use in the DCBA. It allows an ontology designer to design and model an ontology using an UML tool, such as Rational Rose.

However, if an existing ontology is replaced by a new one rather than extended, this approach has implications on the existing knowledge stored in the DCBA. The migration of data in an old knowledge repository to a new ontology model is not supported.

Figure 6.6 represents a UML class diagram of organizational knowledge. The diagram describes a schema for the domain of project knowledge in iBistro.

6.3.2 OBJECT LINKAGE

Named links and version links are similar concepts, albeit solving different needs. Both concepts link a data item to another. Named links denote a relationship be-



1:	An UML design tool, such as Rational Rose, is used to model the ontology. To export the UML model, a XML Model Interchange file (`.xmi`) is exported.
2.1:	The XMI file is transformed to a Java source file (`.java`) using an Extensible Stylesheet Language Transformation (XSLT) stylesheet.
2.2:	At the same stage, the XMI file is also transformed to a RDF schema description embedded in an XML file (`.xml`) by another XSLT stylesheet.
3.{1,2}:	The resulting `.java` and RDF `.xml` files are then compiled into a java class file; the compilation process also relies on the Java RDF API [Melnik, 2000].
4:	The class file is finally instantiated by the DCBA or knowledge sources.

Figure 6.5: Workflow from ontology modeling in UML to concrete knowledge in objects.

tween two objects, or a list of objects, that is not defined implicit by the hierarchical structure of the taxonomy. Version links designate several emerging versions of an object, like a predecessor (older version) and its successors (newer versions). The linkage of objects in the DCBA builds up a complex multi-dimensional network of knowledge interlinked between the different local blackboards.

Named Links

The meaning of a named link is derived from two sources of information. The referenced objects describe *what* is connected and in which direction (the link is directed). The link itself saves information on *how* the relationship is made and *when*. The "how" is a textual description (for instance "created by"), either from a predefined list or user defined, along with a machine readable token. The "when" is defined by a local timestamp. The concept of linking by using objects that can hold additional information on a relationship also enables viewing and browsing the mesh by linking information. It is further extendable (by any attribute) and translates well to the relational model used in the database backend storage.

The concepts of named links and version links can be also combined. A new

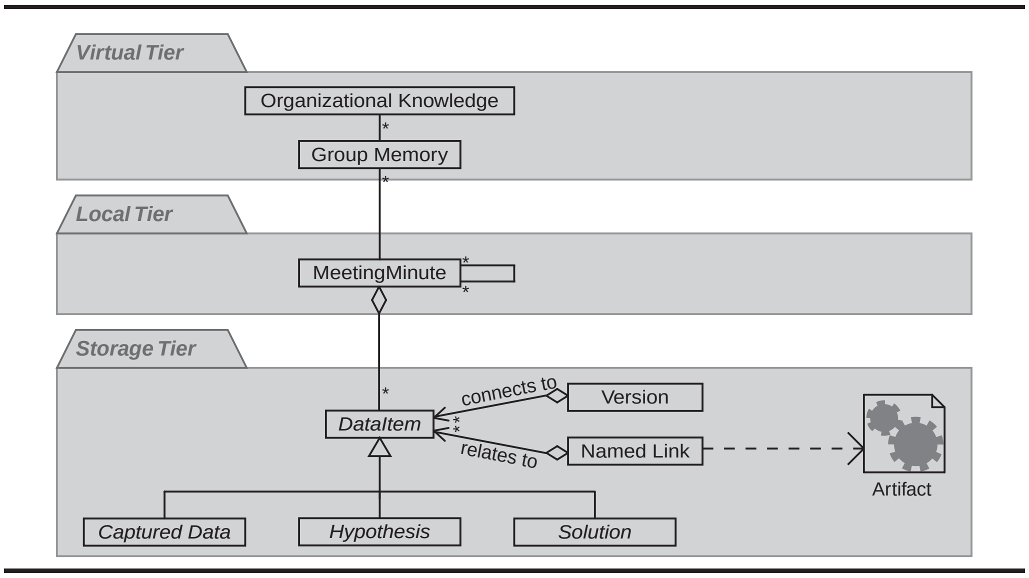


Figure 6.6: The basic model of a project-ontology and knowledge taxonomy for iBistro represented in an UML class diagram. (The complete ontology model is shown in Chapter 7.)

version of an existing object, for instance, likely has a creator, who is designated by a named link.

Version Links

The DCBA control and strategy components typically identify many different and overlapping contexts and foci of attention[4]. In the DCBA knowledge sources are scheduled concurrently. Several instances of the same knowledge source might be executed at a time on overlapping focuses of attention. As a result, knowledge sources work simultaneously on the same or overlapping data items. They further put the result of their work on the blackboard, consequently creating new events that trigger knowledge source activation by enabling the selection of a new focus of attention. This results in a multi-dimensional mesh of data and knowledge.

The potential problems and side effects are addressed by strict version control with the following rules:

- All data in the knowledge space is encapsulated in objects which are readable concurrently by the `get()` operation and its transactions;
- data items cannot be changed. The `change()` operation creates a new version reflecting the changes.

[4]Unlike in the standard blackboard model and due to concurrency of Control classes in the DCBA, there may be several concurrent (and overlapping) *foci of attention*.

- as a result, the `getWait()` operation from the standard blackboard model (Section 2.5) is dropped in the DCBA; (getWait() waits for an object to be released by other knowledge sources and then returns the object.)
- the `take()` operation doesn't actually remove the object.

The strict version control also enables a precise tracking of how the capture and knowledge building process works, and which alternatives are further selected, explored, or dropped.

Two concurrent changes, for instance, will produce two unrelated versions originating from the old version (i.e., a main trunk and a branch or two branches depending how you look at it). The two branches of an older version later can be merged in a new main trunk, one of the versions might be continued, or both are dropped.

The information stored in the blackboard is organized in several layers of abstraction. The lower layers represent lower levels of abstraction, such as raw data. Such data is stored as `Captured Data` in Figure 6.6. Raw data is information that is typically captured automatically, for instance context-events or video, audio, or image snapshots. This information is uninterpreted and potentially not 'understood' by the system for the time being. A whiteboard snapshot, for example, is stored in a bitmap. This bitmap is stored as an object by the `CaptureComponent` in the lowest layer of abstraction. The whiteboard snapshot may contain important information, the meeting's agenda for instance. The captured bitmap, however, is not 'understood' by the system at first. It needs to be interpreted by a knowledge source through knowledge acquisition and then elevated to a higher level of abstraction. In our example, the meeting agenda might be transformed to text, linked with video indices, and stored at a higher level of abstraction (i.e., `Hypothesis` in Figure 6.6). In the blackboard model, every partial solution or intermediate result is called a *hypothesis*[5]. Knowledge in iBistro can also be seen as *explored alternatives* and *selected alternatives*. Explored alternatives, however, are still considered a hypothesis.

6.3.3 PERSISTENT STORAGE

The local blackboards are volatile memory in the runtime RAM storage of each local implementation. In case the blackboard is halted for some reason, its contents are lost. Moreover, complex queries for blackboard items are expensive. Objects in the standard blackboard are found by comparing them against a template object. Thus, objects of a specific type, or with given attributes can be

[5]A *solution* is a finalyzed hypothesis. This means, it will be no further subject to process of knowledge assembly. As a implementation detail in the DCBA, however, there may be several solutions in parallel (*alternatives*), each being a candidate for the selected alternative (final solution).

found. Objects can also be retrieved from the blackboard by their *unique identification* (UID). More complex queries, like searching for parts of names, combinations, links, or intersections of subqueries are not possible. The basic model does also not support complex results for queries, like collections or networks of objects.

To address these problems, the DCBA connects to a relational database system for persistent storage and queries based on the structured query language (SQL). While all data items written to the blackboard are directly written to the database, just like in a caching device using "write-through strategy", queries sent to the DCBA are forwarded to the local blackboards. The local blackboards check for a suitable query result on the blackboard. If no such object is available, they check for the result in the local database. The overall result in the virtual tier of the DCBA is made from the collection of sub-results in the local queries. By this means, the DCBA is filled incrementally with the most current data items from the database. Consequently, the database represents the leading and binding instance in data storage in the overall system. The persistent storage is organized as a replicated database system, as indicated in Step 4.3 in Figure 6.7. The overall result is synchronized and replicated to all inquirers (knowledge sources).

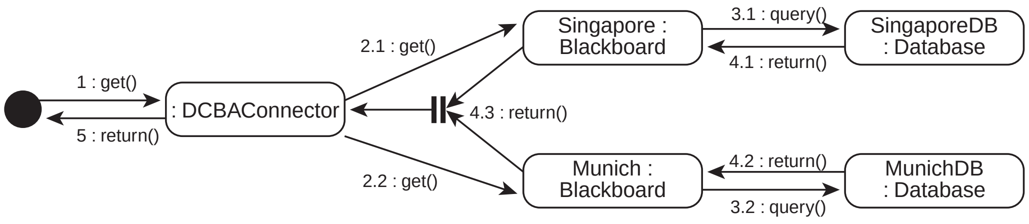


Figure 6.7: Collaboration of components to process a query. The composite result of the query is built from many sub-queries in the DCBA. The figure shows a synchronized call (*call-by-value*). Asynchronous invocation (*call-by-reference*) is possible using Java `RemoteObjects` [Sun Microsystems, 1998]

A query cannot deliver conflicting results from different local blackboards, because every item in the DCBA is under strict version control. This means, that no knowledge item is overwritten or definitely deleted. However, if a specific sub-query delivers an outdated version of a knowledge item, the items are prioritized accordingly to their version number (newest first).

6.4 MEETING CAPTURE

Information capture in iBistro is strictly event-based. Any type of data item in iBistro is stored according to its timely occurrence. Thus, the flow of events in a single meeting follows a common timeline. Incidents captured later, such as the manual post-interpretation of the meeting video, which might result in a (single) requirement, are added with a timestamp representing their post-meeting creation. Surrounding contextual information is linked to the event to indicate the originator (the identity of the person who mentioned the requirement), the time when the requirement was first mentioned, location, and so forth is linked along with the event in the database.

The DCBA connects a many different capture components. Each capture component listens for a specific type of event and forwards it to the DCBA interface. Capture components *actively* propagate the captured data, information, or events to the DCBA. All components have in common that they track events and data that is easily electronically recordable. This spans the possible items into two categories: *context* and *artifacts*.

6.4.1 CONTEXT CAPTURE

An iBistro room consists of many context-aware devices to capture user activity. Context capturing is performed by small active components which propagate any occurring event to the DCBA. Each capture component is typically responsible for a specific type of context. The captured event is written to the local blackboard ("Singapore" in Figure 6.8). The local blackboard writes the event to the persistent storage as described in Section 6.3.3 (this step is omitted in Figure 6.8). Captured meeting events are only propagated to other local blackboards on their first usage.

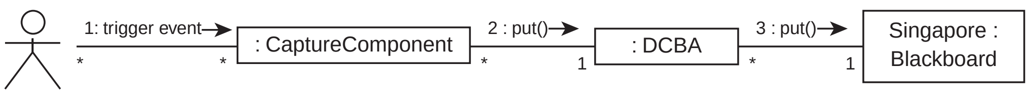


Figure 6.8: Capture of contextual events in the DCBA's `CaptureComponents`.

For instance, a meeting participant using a SMART Board™will trigger an event in iBistro by taking an electronic pen from its pen tray. The pen's color is assigned to the meeting participants (Peter uses green color, Florian the red pen, and so on). At the context capture stage, the sensor only knows about the event (taking the green pen). The semantic interpretation of the event is performed by a knowledge source (this is actually also a simple form of knowledge acquisition by translating the lower-level information"green pen taken" to the higher-level knowledge of "assigning a sketch to Peter being the actuating stakeholder".)

The capture component propagates the event to the DCBA and triggers notification. The capture component can be both an active or passive component which either repeatedly checks for events or gets notified.

6.4.2 ARTIFACT AND VIDEO CAPTURE

By *artifacts* in the iBistro model, we understand software artifacts used in the development process. Artifacts are typically files containing source code, documentation, etc. In artifact capture, events are triggered by changes of monitored software artifacts. The main difference to context capture is a technical rather than a conceptional limitation: While a 'change-event' is generated and propagated to the DCBA, the artifact itself cannot be directly stored in the DCBA for performance and legacy reasons[6] and due to storage space available[7].

If an artifact capture component, for instance, monitors a source file, the source file cannot be transferred to the repository each time it changes. During development, the programmer likely changes (and saves) the file over and over again. The version checked into the version control system, however, must have a defined status (such as *under work, fails to compile, tested, ready to deploy, ...*). All changed artifacts thus need to be accessible by the system to save the change history. Hence, the artifact capture components have to collaborate with specific second level storage systems, for instance a web-server reference specified by an URL, a local or remote file identified by the filesystem path or remote access path, or a version control system such as the Concurrent Version System (CVS).

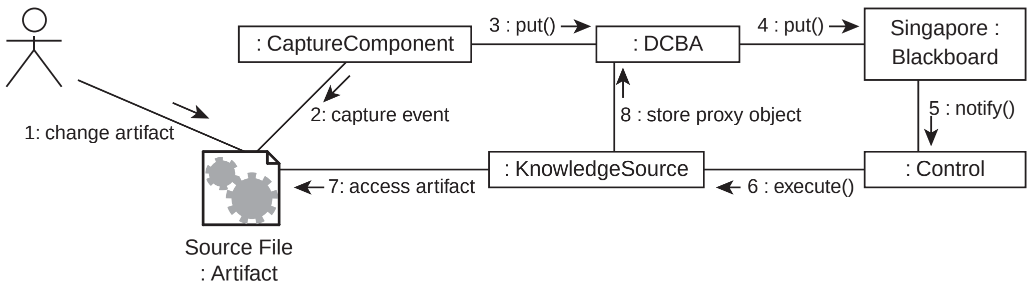


Figure 6.9: Capture of artifacts and video streams and knowledge source scheduling as a consequence (collaboration diagram).

Figure 6.9 displays capturing of artifacts that are under version control, for instance CVS, in the DCBA:

1. User checks out an artifact (for instance source code). The User works on the artifact and successively saves changed versions to the file system for compiling and testing. Finally, the user declares a version as *under work* and checks in that version of the artifact.

[6]Legacy compilers, text editors, and IDEs will operate on source code files and not on a DCBA.

[7]Object space for DCBA storage is limited to approximately 5 megabytes per object with the current default Java settings. While larger objects could be stored easily in a changed configuration, the storage of large objects in the DCBA proofed neither efficient nor suggestive. See Chapter 7 for details on the prototype implementation.

2. A specific tracker captures the event of a changed source file.

3. The change event is written to the DCBA.

4. The event is written to the affected local blackboard (i.e., at the client-site).

5. The blackboard notifies control of an artifact change.

6. Control schedules a knowledge source for artifact capture and generation of a proxy object.

7. The knowledge source checks the artifact, for instance by accessing it using the version control system. (The knowledge source does not need to check the respective proxy object in the blackboard explicitly, as it is part of the focus of attention which is sent to the knowledge source at its invocation.)

8. The proxy object which represents the artifact in the DCBA is generated and stored.

6.5 KNOWLEDGE ACQUISITION

The underlying concept behind knowledge acquisition is to transform information that is already there in principle to a usable (and understandable[8]) form. The term knowledge acquisition is derived from knowledge management to denote the fact that, while information is available within an organization, it might not be usable. In an organization, specific categories of knowledge can only be properly used by specific domain experts, for instance. Every so often, several domain experts need to collaborate on a set of data to collectively assemble partial solutions and hypotheses to find a resolution.

This process is modeled in iBistro. Specific knowledge sources work on well-formed categories of data items. Knowledge sources are very flexible and can be simple rules or very complex, or anywhere in between. The term knowledge source refers to the original idea that each knowledge source contributes a bit of knowledge to the solution of the problem. From a software engineering point of view, knowledge sources allow for the modularization and composition of the iBistro system.

[8]The term knowledge acquisition is used both with the notion of human understanding and making s.th. readable or interpretable for machines. A useful generalization used in this context in knowledge management is that machines *know* and humans *understand* [Gordon, 1999].

6.5.1 MANUAL VS. COMPUTATIONAL KNOWLEDGE ACQUISITION

The DCBA architecture supports two generic types of knowledge sources[9]. The first are *computational* knowledge sources which are triggered by an event and work on objects from the repository automatically and self-sufficiently. The second category are *manual* knowledge sources which provide a user interface for knowledge acquisition to a human user. An example for the first category, computational, is a group recognition service which identifies a specific work group by the presence of the individual group members. An example for the second category (manual knowledge source) is the MEETINGGENERATOR tool, which is scheduled after a meeting and is used by a human meeting champion for post-mortem editing and authoring of the meeting.

6.5.2 KNOWLEDGE SOURCES

Knowledge sources (see Figure 6.2), in contrast to capture components, are *passive* components in the DCBA. Knowledge sources are scheduled and executed by the `Control` class. If evaluation of a knowledge sources' *precondition* is true, the knowledge sources' code (*body*) is executed. The evaluation whether to execute the knowledge source is done using the object or objects that caused the notification as input parameter(s) to the precondition. While the evaluation of a knowledge source is triggered by the Control class, the evaluation code (the precondition) is embedded in the knowledge source's precondition. The knowledge source typically creates a resulting object either on the same level of abstraction or one level higher.

The Knowledge Source Precondition

A knowledge source's precondition is a rule, expressed in a function. The precondition is part of the knowledge source and is scheduled along with the focus of attention. As the DCBA is a concurrent system, all knowledge sources with satisfied preconditions on the given set of objects are ready for execution. This behavior is similar to the Official Production System 5 (OPS 5) rule-based system [Forgy and McDermott, 1977]. In the DCBA, the knowledge sources containing the preconditions are bound with their context (related objects) in the focus of attention. The preconditions in the focus of attention are then evaluated. The focus of attention is spent after usage and is re-built in each control cycle. All preconditions are re-evaluated in each control cycle.

All knowledge sources remain enabled until they are stopped explicitly, even though the state of the blackboard changes. The potential constraints which typ-

[9]The History class shown in Figure 6.2 is omitted here, because it is only used for internal logging and research purposes.

ically arise from this do not apply in the DCBA due to the strict version control: the context of execution for one specific knowledge source remains the same during his execution, at least for one specific instance of the knowledge. The knowledge source, however, might continue exploring an alternative which gets outdated during execution of the knowledge source.

As knowledge sources in the DCBA are executed in parallel, the precondition is a boolean function. Strategy, however, may decide whether to actually start a knowledge source if its precondition returns true based on global information, for instance system workload.

The Knowledge Source Body

The *body* is the executable knowledge generating portion of the knowledge source. The body can be any kind of executable code, for instance embedded code or external programs and scripts. The body is not restricted to the context or focus of attention used during knowledge source scheduling and in the precondition as source of data. The body has reading access to any portion of the local or virtual global blackboard (the `DCBAConnector`) and can access all objects that were already there at the time of the knowledge source's invocation (identified by the time stamps). In practice, however, the body will likely work only on a small portion of the blackboard and likely with other sources, such as local information. The results of the computation are put to the blackboard. In bottom-up reasoning, the result is put one abstraction layer above the layer data is taken from. In top-down reasoning, in contrast, the result is put to the layer just below the source layer.

The body is not restricted to accessing context information in the DCBA. It can also access artifacts, which are stored externally via the proxy objects in the DCBA. Communication between knowledge sources, however, is restricted to using the blackboard as an infrastructure. The DCBA as the common repository also offers a means for knowledge source's body to store internal state information on the blackboard for later usage (for instance learning), or persistent storage.

6.5.3 KNOWLEDGE SOURCE SCHEDULING AND EXECUTION

Figure 6.10 shows the procedure of knowledge source scheduling and execution. If the state of the blackboard changes, the local blackboard notifies the DCBA which notifies central control. The Control class then uses the affected item or items to select the focus of attention, which is a collection of blackboard objects (from any local blackboard, but available at the virtual layer) and knowledge sources. As this collection is identified at the broker tier, it might span several objects and knowledge sources on distributed sites (i.e., the client-site and the offshore development site). To further identify which foci of attention are to be executed, the current instance of the strategy component is used for evaluation. A very simple strategy class might simply call each knowledge source's precondition and add the knowledge source to the scheduling list if it returns 'true'. A more complex Strategy class might weigh the results of the preconditions and schedule the knowledge source in a defined order according to the weighting. Finally, the generated scheduling list of knowledge sources is executed concurrently with their according focus of attention (which still contains the relevant data items).

```
while( _bb_state == changed ) {
  collection += selectFocusOfAttention(notification.getNextEvent());
  schedule   += strategy.evaluate(collection);
}
for(i=0; i<schedule.length(); i++) {
  schedule.get(i).body.run(schedule.getCollection[i]);
}
```

Figure 6.10: Knowledge source evaluation and scheduling.

6.6 KNOWLEDGE VIEWS AND RETRIEVAL

The data, information, and knowledge stored in the DCBA are nodes in large graphs of interconnected data items and other objects representing meetings and information in its timely order. It includes external annotations from other sites or an individual's personal computer. Knowledge retrieval gathers the contents and structure of the knowledge and information stored in the DCBA and provides a application specific view on that knowledge. During knowledge retrieval the users need different views that only include the relevant details that are useful for the current task. Presenting the knowledge gathered during a meeting in a chronological order might be useful during the postprocessing. Presenting the same knowledge focusing on the decisions that were taken might be useful for establishing the context for the next meeting.

Like in the model-view-controller pattern various *knowledge views* provide different visualizations of the state of the repository, information, and structure of a knowledge space. Knowledge views are used in particular to provide an interface for human users of the system.

Accordingly, the knowledge space can be seen in its timely occurrence (*temporal views*), by browsing it based on various key of search, or by following the links which connect the data items.

6.6.1 TEMPORAL KNOWLEDGE VIEWS

A self-evident way to view a meeting is to playback the meeting as a multimedia archive, thus enabling non participants to access the raw information. In iBistro, the MEETINGVIEWER generates on-demand a SMIL [W3C, 1998a] file (or data stream) to represent the meeting along with the captured requirements, context, rationale, and so on. This allows interested people to navigate through a meeting using any SMIL compliant video player, such as RealPlayer^{TM} or Quicktime's Movie Player to view the meeting. As the content of the meeting follows a common timeline, a "clip position" slider is used to navigate through the captured audio, video, as well as other content such as requirements. Alternatively, the history events can be used to jump to specific segments of the meeting minutes, for example, navigating an option will move the position slider to the frame where the option was first suggested. Graphical views of requirements or rationale can be displayed using HTML or by generating bitmaps on demand. However, displaying multi-dimensional components, such as context-links between stored entries which allow navigation, is not possible in this simple view..

6.6.2 MEETING BROWSERS

As knowledge in iBistro is stored along with its related contextual information, navigation is possible using various types of input (or keys in a search). Meeting minutes consist of contextual information (e.g., location, identity, activity, history, and time) which can serve as keys for searching. For example, meeting minutes may be sorted by requirements authored by a certain participant, by time, or on any other key. Navigation is possible on any of those keys: the stakeholder of an issue is found by clicking on that issue. Related information, like time or location where the meeting took place, is displayed accordingly and might be used for further navigation. Thus, iBistro's database can be used to find stakeholders over various meetings or even projects. While a MEETINGVIEW provides a meeting-based index into the knowledge base, other knowledge sources can provide an artifact-based view into the knowledge base.

As currently implemented in the MEETINGGENERATOR, the knowledge base must be searchable by any type of context, e.g., by stakeholders, location, topic, versions, and so on. Hence, in addition to the raw context information captured during the meeting, the user also sees all the annotations and structure that were added during the post-processing.

6.6.3 3D KNOWLEDGE VIEWS

The post-mortem process generates interlinkage of knowledge stored in the repository. This information can be translated to a three-dimensional model as shown in Figure 6.11.

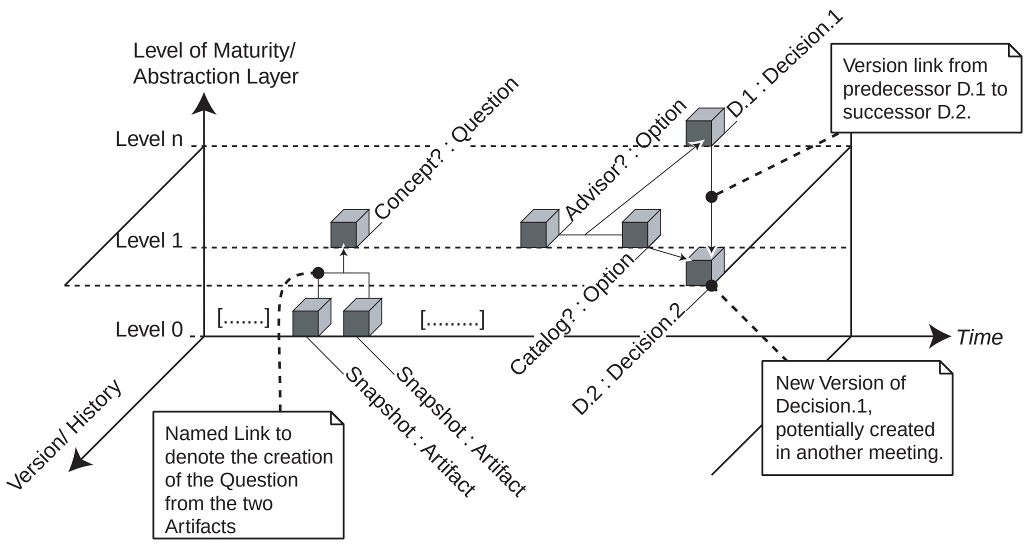

Level of Maturity/ Abstraction Layer
Level n
Level 1
Level 0
[.......]
[.........]
Time
Version/ History
Concept? : Question
Advisor? : Option
Catalog? : Option
D.1 : Decision.1
D.2 : Decision.2
Snapshot : Artifact
Snapshot : Artifact
Version link from predecessor D.1 to successor D.2.
New Version of Decision.1, potentially created in another meeting.
Named Link to denote the creation of the Question from the two Artifacts

Figure 6.11: Information in iBistro seen as a 3D-model of knowledge.

The three axes represent the timeline (x-axis), the level of abstraction (represented in the blackboard layer, y-axis), and version or knowledge-interlinkage (z-axis).

For example, one could modify the REQuest [Dutoit and Paech, 2001a] requirements engineering tool so that developers can browse meeting segments based on a specific scenario or use case. By providing a seamless integration between meetings, models, and documents, the value of iBistro will be more visible to the meeting participants.

6.7 SYSTEM SUMMARY AND PROPERTIES OF THE DCBA

Figure 6.12 shows the overall iBistro system with the DCBA as a virtual tier which is built from the two local iBistro sites, Munich and Singapore. The figure shows all components in a global iBistro system built on the DCBA.

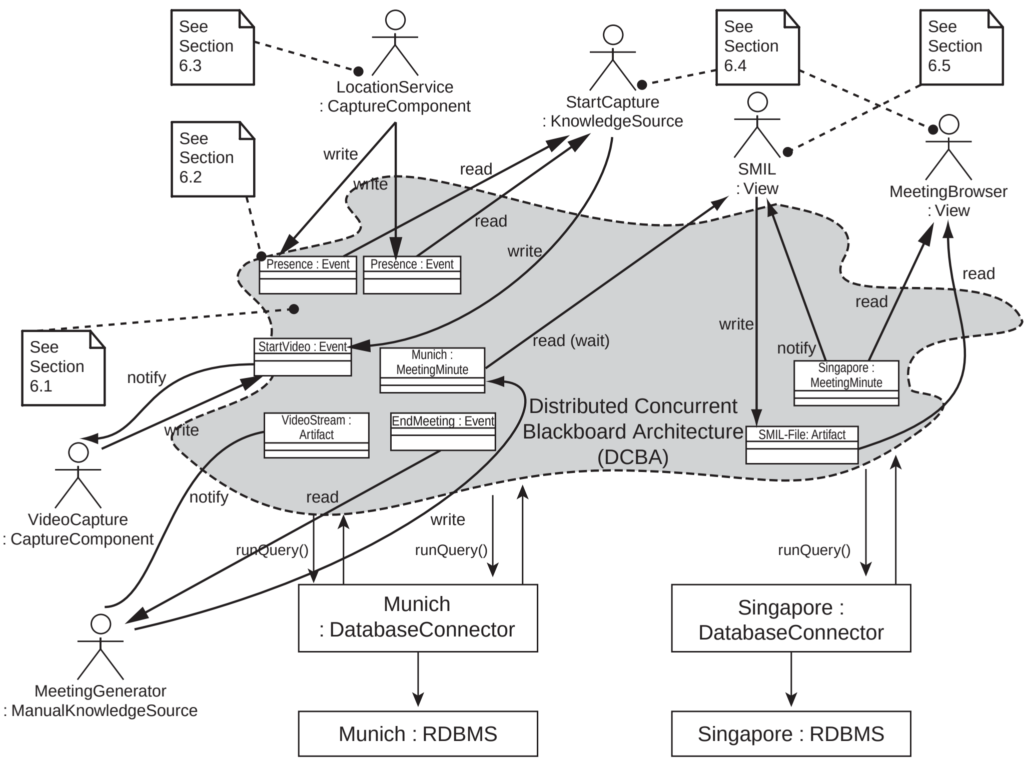


Figure 6.12: Overview of the distributed concurrent blackboard architecture used in iBistro. The DCBA blackboard realizes transparency for both users and components used in the system, such as knowledge sources. The actual local blackboards are omitted in the figure.

6.7.1 TRANSPARENCY

The DCBA is a virtual connector for users and components (KnowledgeSources, CaptureComponents, and Views). It is built up from local blackboards which serve as a caching and data storage middleware. Regardless of the actual location of the server a component connects to, the content and functionality of the communication infrastructure and repository is the same. The DCBA provides *transparency* by hiding the underlying technical conditions. This is contrary with existing blackboard systems. As iBistro is a globally distributed system, components would have to be aware of the location of the system and the knowledge state of the system they are connecting to. For instance, if a user in Singapore needs to access a document from Munich, she would have to know about the existence of the Munich document (or find out by searching) and connect to the Munich repository. In a transparent system, in contrast, this process is hidden in the architecture without being apparent to users and connected components.

6.7.2 THE EFFECTS OF SCALE

In the standard blackboard model, knowledge sources are not arbitrary accessing the blackboard due to the strict and non-concurrent scheduling. In the DCBA, in contrast, distributed components are free of choice. Especially the CaptureComponents do what they want, when they want, and make their decisions without knowing anything about the other components except for the results they produced so far. Nevertheless, they appear to be *scale effective* [McManus, 1992]. An organization of software components is scale effective if its performance improves with size. An organization based on the distributed concurrent blackboard architecture is scale effective if there are components whose addition improves the quality of solutions or systems whose addition improves solution-speed.

Scale effectiveness is an desirable property for distributed systems. The problem of improving the performance of a scale effective organization reduces to the problem of finding which components to add. A non-scale effective organization faces the much more difficult problem of finding which of its parts to eliminate or modify before additions can be of benefit. Synthetic organizations are often scale ineffective. The proverb, “too many cooks spoil the broth” describe this situation. That is, if some size is reached, the addition of another “cook”, no matter how competent, has a negative impact on overall performance. In a scale effective organization there can never be too many “cooks”, as long as it retains it’s property of being scale effective.

The DCBA is scale effective because adding new local blackboards scales up the performance for larger teams, more sites, or more teams. A greater amount of teams is then able to accomplish more complex projects. The quality of solutions or the difficulty of solved problems will not be affected. However, the addition of improved knowledge sources and components can increase the level of team support. In the next chapter we will have a discussion of performance of the DCBA compared to a single blackboard system.

6.7.3 COMPARISON WITH THE ORIGINAL BLACKBOARD METAPHOR

In contrast to the generic blackboard model or other blackboard-based systems, our system embodies several specialities:

- Knowledge sources work *concurrently* on the DCBA and even on the local blackboards. The system, however, needs no locking, such as mutexes or semaphores (except for some atomic operations) on the blackboard. Every knowledge source that works on the same data item never changes this item, but generates *new versions* which are derived from the ancestor.
- As a result, each layer stores several *concurrent versions* of knowledge objects. This also implies that, unlike other blackboard systems, data items that were already used are not deleted from the blackboard. Hence, the complete history of knowledge assembly is stored in the blackboard. As an implementation detail, versioning is handled within the layers (Class `AbstractionLayer` in Figure 6.2).
- To support the empirical evaluation capabilities of iBistro, the DCBA uses logging entries created by the history component to store log files and rationale information on knowledge source activation (such as which rules triggered knowledge source activation on a certain object) and event logging for both contextual events that occur during the meeting and event and object notification within the blackboard system. This information is also needed to control the advancement of finding solutions within the system, as used objects are not deleted from the blackboard. Logging is triggered directly by the `Control`-class. (See Figure 6.2.)
- Data items and artifacts are stored in a relational database. This allows persistent and efficient storage of data items and powerful item interlinkage (relations) and queries. The main advantage of relational data storage is the persistent storage of knowledge, in contrast to the non-persistent storage in the blackboard layers. This enables failover robustness for the blackboard.
- The second reason for using a relational database is that the structured query language (SQL) can be used as a standardized query language for knowledge sources and other components.
- One single DCBA installation, for example in Munich, interfaces other servers (e.g., Singapore) by implementing the `DCBAConnector` class shown in Figure 6.2. The overall DCBA is built from several server's connected DCBA interfaces. Any `DCBAConnector` serves as a single point of contact for any knowledge source, hence providing *transparency*.

In this chapter we discussed the design and details of the iBistro architecture, the distributed concurrent blackboard architecture (DCBA). In the next chapter, we introduce several actually implemented prototypes and tools and related case studies and tests.

CHAPTER 7

EMPIRICAL APPROACH AND RESEARCH PROTOTYPES

Inside every large program is a small program struggling to get out.

[HOARE'S LAW OF LARGE PROGRAMS]

The iBistro system was developed incrementally. Figure 7.1 summarizes the timeline for implementation, testing, and experimentation. This chapter illustrates the iBistro system and test plan used during three iterations.

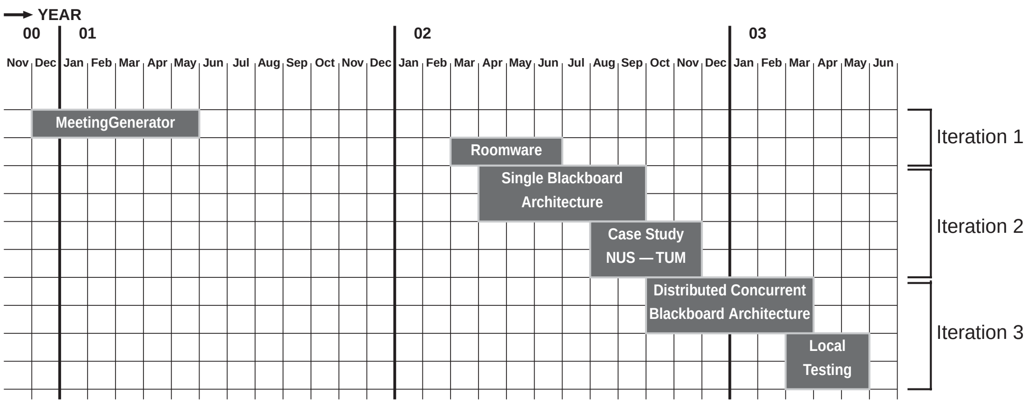


Figure 7.1: Overview of the incremental development and case studies.

The focus of the first iteration was the development of the MEETINGGENERATOR tool to capture informal meetings in annotated video and audio. In the first iteration, SMART Boards were used during informal meetings and especially in post-mortem processing the meeting with the MEETINGGENERATOR. The

SMART Boards also allowed for collaborative usage of the iBistro tools in small local workgroups and in seminars.

The focus in the second iteration was the development of the single blackboard architecture to prepare the MEETINGGENERATOR and iBistro for an actually distributed project between TU München (TUM) and the National University of Singapore (NUS). In the third iteration, based on the experiences with the single blackboard architecture, the distributed concurrent blackboard architecture was developed as an improved version. This version of the system was tested locally at Munich using a test harness.

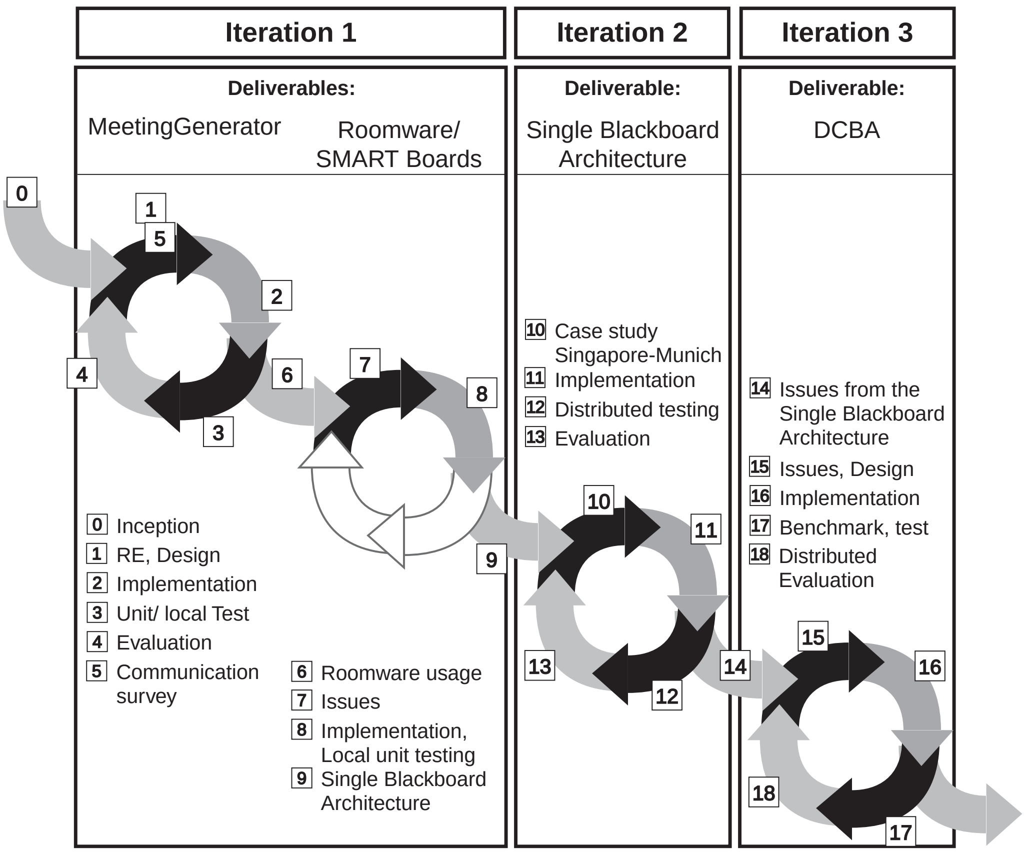


Figure 7.2: The three iterations and resulting deliverables during iBistro implementation and evaluation.

The iterative approach with the individual steps for the three iterations to produce the respective deliverables (the MEETINGGENERATOR, the support for roomware/ SMART Boards, the Single Blackboard Architecture (SBBA), and the Distributed Concurrent Blackboard Architecture (DCBA) is shown in Figure 7.2. In the following sections, we will detail the iterations and their associated deliverables.

7.1 ITERATION 1: THE MEETINGGENERATOR

The MEETINGGENERATOR [Braun et al., 2001a; Hengstenberg, 2001] provides a meeting recording and editor tool to capture informal team meetings and related contextual information in XML annotated audio and video. After the meeting, the recorded video stream is edited by the *meeting champion* who is assisted by the editor component in the tool. The meeting champion is a meeting participant (for instance, a consultant or team-member), who is preferably not a meeting facilitator.

The MEETINGGENERATOR uses a rationale-based approach to structure an informal meeting accordingly to *questions* that come up or are discussed during the meeting, *options* that suggest alternatives and ideas to address the questions, and *criteria* against which the options are evaluated by *assessments*. Figure 7.3 shows a QOC table as used in the meeting generation. The approach used is derived from QOC design space analysis [MacLean et al., 1996].

Question	Criterion C_1	Criterion C_2
Option O_1	+	o
Option O_2	–	–

Figure 7.3: A QOC diagram. The options O_1 and O_2 are assessed against criteria C_1 and C_2. The assessments in this example are '+' (for "pro"), '–' (for "con"), and 'o' (for "neutral").

7.1.1 SCOPE

Figure 7.4 shows the model of a meeting in iBistro. Meetings in iBistro, in contrast to impromptu meetings and hallway conversations, are work meetings with an expected project or business impact and outcome. Due to the result-oriented character of meetings in iBistro, the post-mortem structuring of captured knowledge is crucial. In general, this takes place after the brainstorming session. The post-meeting structuring is typically performed by the meeting champion.

During the post-meeting, the meeting champion annotates the captured audio and video stream with higher-level information to provide an index into the raw material. In iBistro, we use the question, option, criteria paradigm as a basis for these annotations. The meeting champion identifies topics that were discussed by attaching a Question event to a segment of the tape. Within that segment, the meeting champion identifies different alternatives with different Option events. The meeting champion documents decisions, such as the selection of an alternative or the discarding of others, by creating a Decision event. Since History events, like all other iBistro events, have an attribute identifying the originator

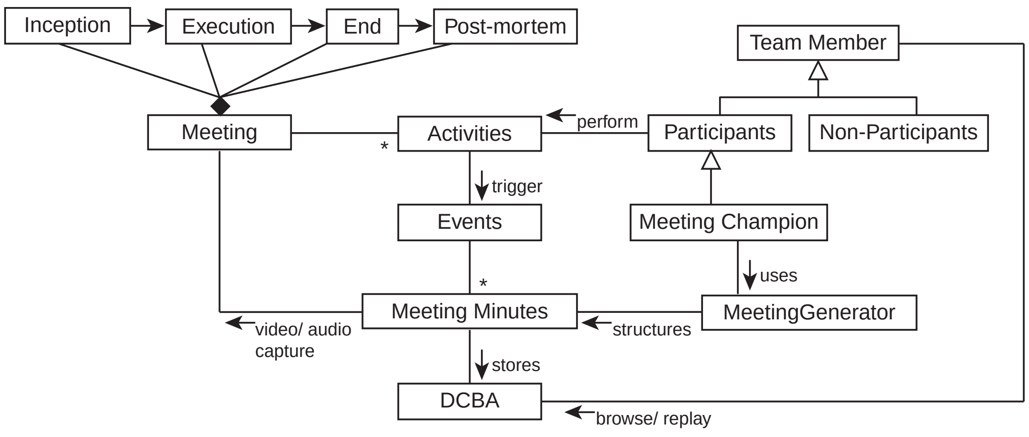


Figure 7.4: The model of a meeting in iBistro (UML class diagram).

of the event (in this case the person who suggested the option or who made the decision), traceability to human sources is ensured. In addition to history events, the editor can also attach other types of information using Link events, such as references to other material, for example, a problem statement from the client, scenarios, and questions generated by REQuest[1], or a class diagram generated by a CASE tool.

7.1.2 APPROACH

The MEETINGGENERATOR supports the meeting champion during the meeting and in post-mortem editing of the meeting minutes. The meeting champion is in a similar role to a minute taker. The meeting champion, however, does not need to take notes during the meeting.

Figure 7.5 shows the roles of the meeting champion and meeting participant and displays the basic use cases for the MEETINGGENERATOR tool. During the meeting, the meeting participants trigger context events (for instance by entering or leaving the room, using electronic whiteboards etc.) which are forwarded to the capture event use case. The meeting champion directs the recording of the meeting minutes and may capture additional events for which no automated capturing mechanism exists, such as adding markers for important milestones of the meeting. Context-events captured automatically may also trigger events in the MEETINGGENERATOR. For instance, using the electronic whiteboard starts minute capturing, while switching off the whiteboard stops the electronic minute recording. Figure 7.6 displays the use cases during meeting recording.

[1]REQuest [Dutoit and Paech, 2001a] is a Web-tool for rationale-based use case specification. REQuest enables users propose requirements and their justifications, to review and to discuss

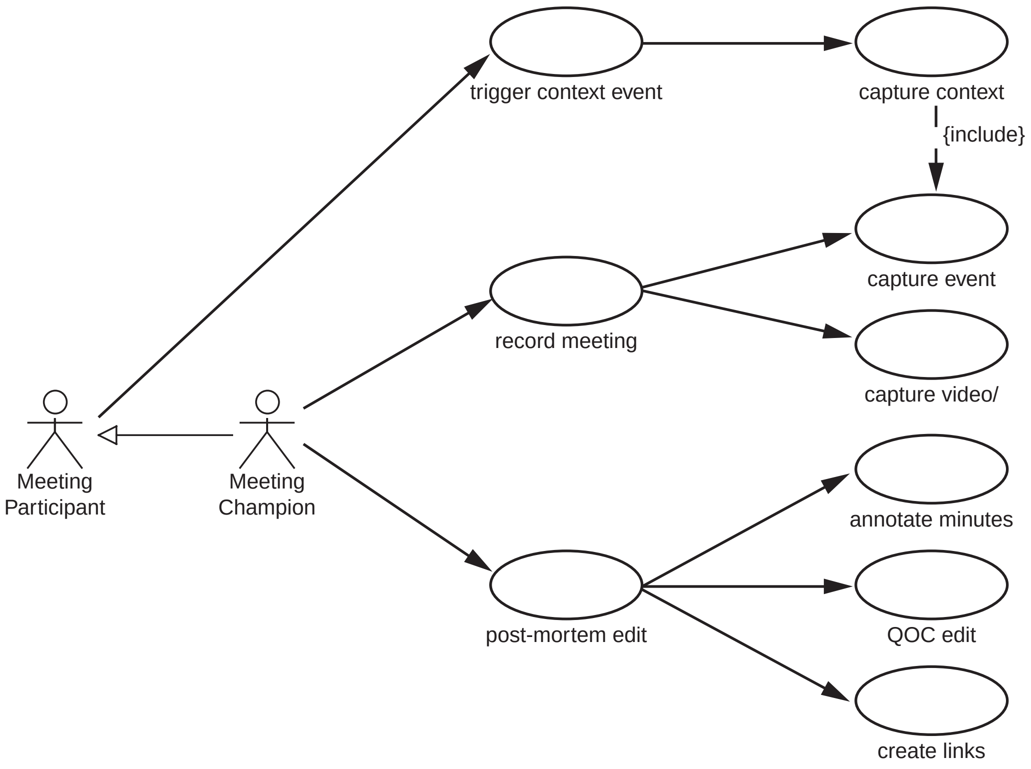


Figure 7.5: Actors in the MEETINGGENERATOR tool (UML use case diagram).

Figure 7.8 shows the MEETINGGENERATOR tool in recording mode. The left window shows the control panel and the meeting minutes, represented as a list of captured events and knowledge items. The right window shows the video frame with the current camera view.

During the meeting, the lower minutes panel is filled with context events. Context events can also be created manually by the meeting champion using "Add un-/named Mark/ Topic" to add a marker (add marker use case).

"Simulate Context Event" manually creates a context event which is added to the meeting minutes. This is used during experimentation or testing when only few automatic context sensors exist. For instance, an event for a person joining the meeting can be created manually if no active badges location tracking is available. After the meeting is ended, the meeting champion stops the meeting capture ("Stop Minute"). This brings the MEETINGGENERATOR in editor mode for minute post-processing and QOC-annotation.

In editor mode, the MEETINGGENERATOR is used to post-process and structure the meeting minutes after the meeting participants have left the room. Figure 7.7 shows the use cases for the post-mortem process after the actual meeting.

them using the QOC paradigm.

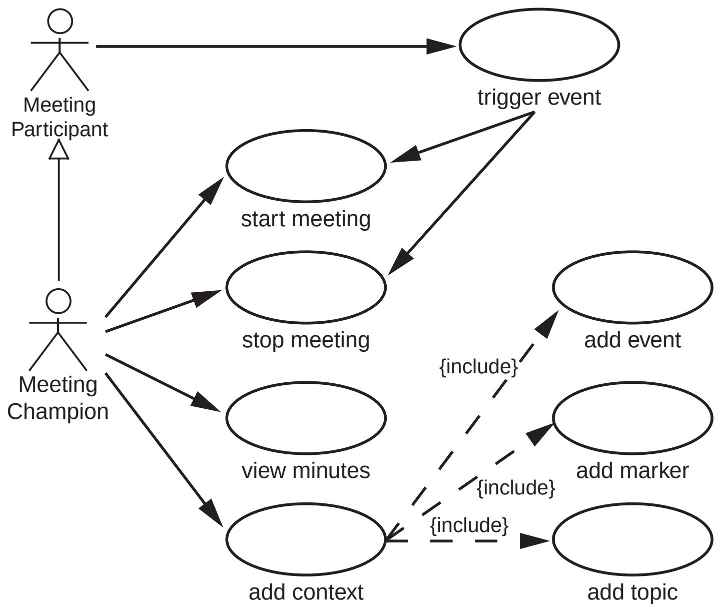


Figure 7.6: The record meeting use cases for meeting recording (UML use case diagram).

After the meeting, the meeting champion edits (post-mortem edit) the recorded meeting minutes by annotating (annotate minutes), structuring (QOC edit), and linking (create links) the captured context and information. At the beginning of the post-mortem process, the captured meeting minutes which contain the audio and video of the meeting and captured events (either automatically captured or added manually) can be browsed (navigate minute). The content is then edited in two ways: post-mortem context can be added after the meeting with hindsight (for instance adding question, option, criteria, or comment tags) or the existing content can be linked (for instance by linking questions to stakeholders).

Figure 7.9 shows a screenshot of the MEETINGGENERATOR in editor mode. The video player window displays the video stream of the meeting and plays back the meeting audio. By clicking on a meeting event in the meeting minute window, the video player window will display the corresponding video position. By selecting a video position in the video player window using the timestamp slider, the meeting minutes will be displayed accordingly.

Contents of the SMART Boards can also be linked with the video/ audio stream. This is achieved by linking an event or higher-level object representing the SMART Board content with the video.

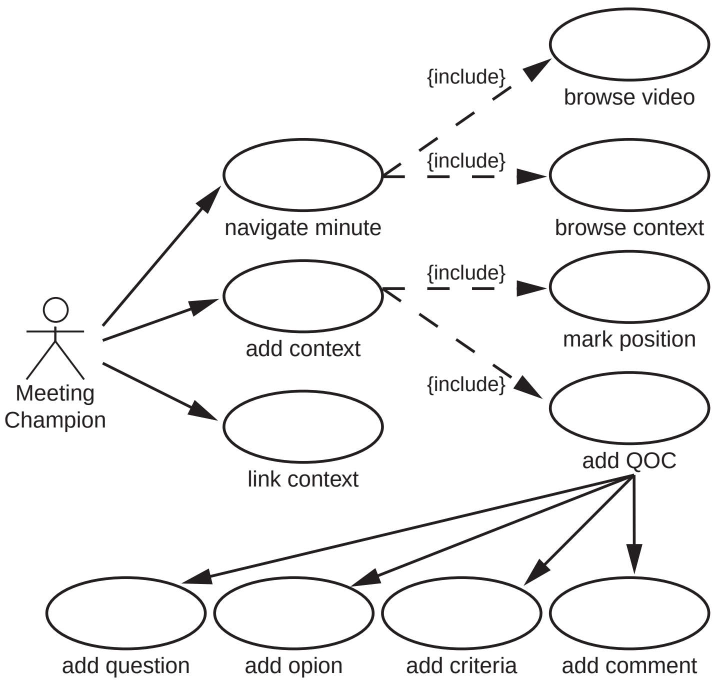
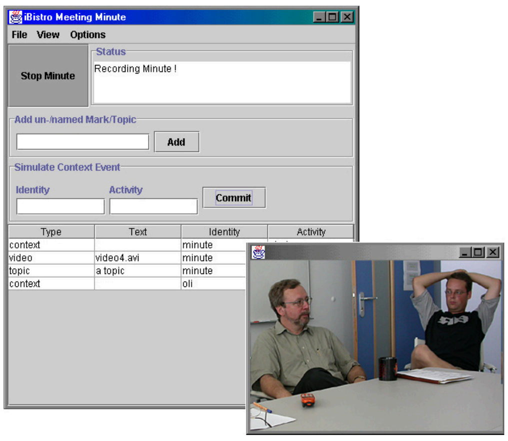


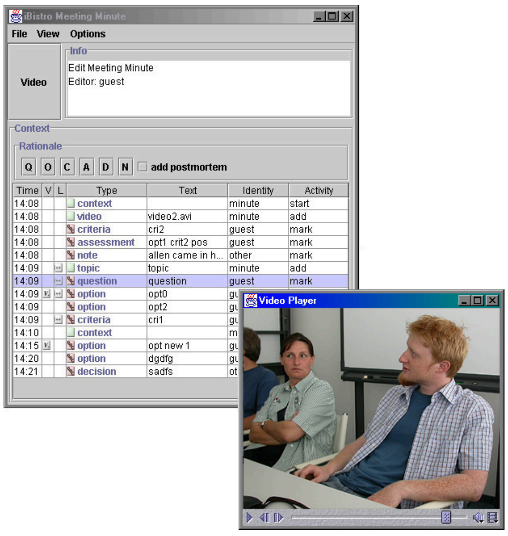
Figure 7.7: The record meeting (UML use case diagram).



Recording Minute !

Type	Text	Identity	Activity
context		minute	
video	video4.avi	minute	
topic	a topic	minute	
context		oli	

Figure 7.8: A screenshot of the MeetingGenerator in recording mode.



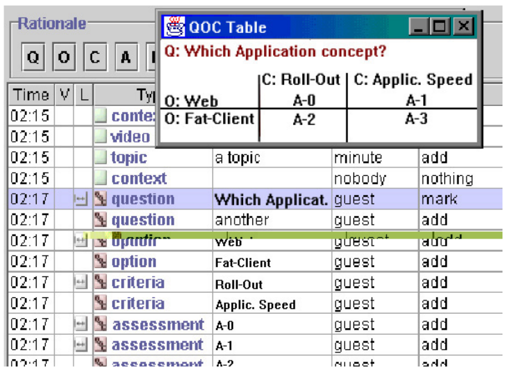
Edit Meeting Minute
Editor: guest

Time	V	L	Type	Text	Identity	Activity
14:08			context		minute	start
14:08			video	video2.avi	minute	add
14:08			criteria	cri2	guest	mark
14:08			assessment	opt1 crit2 pos	guest	mark
14:08			note	allen came in h...	other	mark
14:09		↔	topic	topic	minute	add
14:09		↔	question	question	guest	mark
14:09	✓	↔	option	opt0	gu	
14:09			option	opt2	gu	
14:09		↔	criteria	cri1	gu	
14:10			context		m	
14:15	✓		option	opt new 1	gu	
14:20			option	dgdfg	gu	
14:21			decision	sadfs	ot	

Figure 7.9: A screenshot of the MeetingGenerator in editing mode.

For each question in the meeting minutes, an QOC table view can be displayed (Figure 7.10) by clicking with the right mouse button on the question. The QOC table displays the options, criteria, and assessments linked to a question. Further links can be linked to a question by dragging and dropping options, criteria, and assessments into the QOC table window.

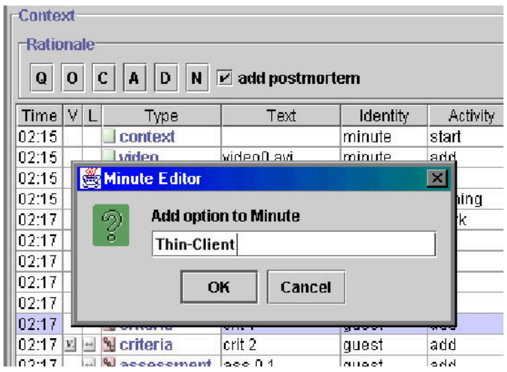
Using the editor mode, the meeting champion is able to add QOC events using the Buttons Q, O, and C, as well as assessments (the button labeled "A"), decisions (Button "D"), and to take notes (Button "N"). Figure 7.11 shows how an option event is added. The timestamp (leftmost column in the event table) is taken from the meeting video's current time position by default. The "add postmortem" box can be used to add information that is not taken directly from the meeting video. If the checkbox is activated, the added event is created with the post-meeting timestamp add linked to the actual video position.



Q: Which Application concept?

	C: Roll-Out	C: Applic. Speed
O: Web	A-0	A-1
O: Fat-Client	A-2	A-3

Time	V	L	Ty			
02:15			conte:			
02:15			video			
02:15			topic	a topic	minute	add
02:15			context		nobody	nothing
02:17			question	Which Applicat.	guest	mark
02:17			question	another	guest	add
02:17			option	Web	guest	add
02:17			option	Fat-Client	guest	add
02:17			criteria	Roll-Out	guest	add
02:17			criteria	Applic. Speed	guest	add
02:17			assessment	A-0	guest	add
02:17			assessment	A-1	guest	add
02:17			assessment	A-2	guest	add



Add option to Minute

Thin-Client

Time	V	L	Type	Text	Identity	Activity
02:15			context		minute	start
02:15			video	video0.avi	minute	add
02:15						
02:15						ing
02:17						k
02:17						
02:17						
02:17						
02:17						
02:17			criteria	crit 1	guest	add
02:17			criteria	crit 2	guest	add
02:17			assessment	ass 0.1	guest	add

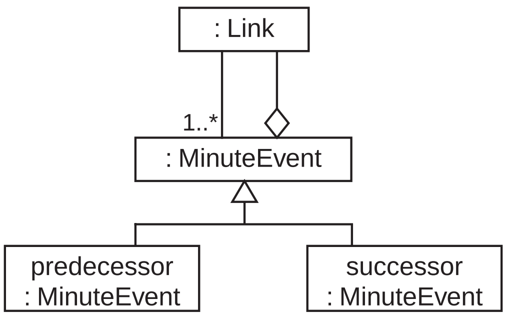
Figure 7.10: View the QOC representation of the meeting.

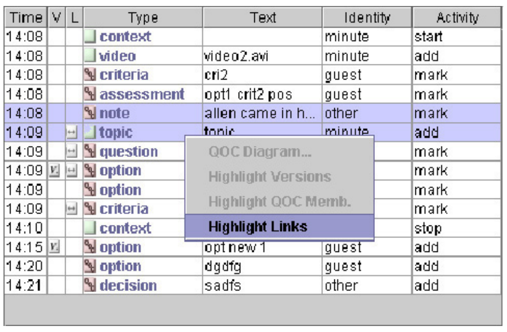
Figure 7.11: Add a QOC-option to the meeting minute.

Events and knowledge items in a meeting minute can be linked by three types of links. *Generic links* connect two items, for instance a bitmap to its originator. *QOC links* connect items which together build a QOC table. *Version links* connect the initial version of a knowledge item with all its successors.

Figure 7.12 shows the highlighting of generic links in the meeting minutes event list. Generic links connect related context-events or knowledge items in the meeting minutes list. As shown in Figure 7.13, a predecessor can be linked to several successors on the same level. Each successor can be linked to it's successors in turn.

New versions are added to existing minute entries by choosing "Revise Entry". Each revision will automatically create a new version linked to its originator. Figure 7.14 shows the creation of a new version of an existing criterion. In the example, the criterion labeled "Roll-Out" is changed to "Deployment: Training", for instance regarding the fact that the roll-out phase was discussed in several in-



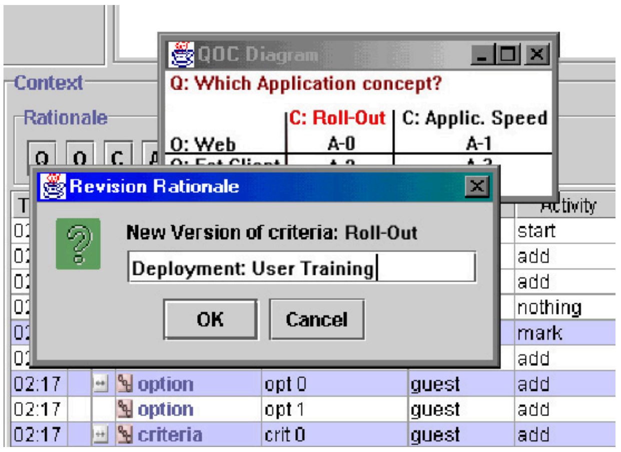


Time	V	L	Type	Text	Identity	Activity
14:08			context		minute	start
14:08			video	video2.avi	minute	add
14:08			criteria	cri2	guest	mark
14:08			assessment	opt1 crit2 pos	guest	mark
14:08			note	allen came in h...	other	mark
14:09			topic	topic	minute	add
14:09			question			mark
14:09			option			mark
14:09			option			mark
14:09			criteria			mark
14:10			context			stop
14:15			option	opt new 1	guest	add
14:20			option	dgdfg	guest	add
14:21			decision	sadfs	other	add

Figure 7.12: Meeting minute navigation by highlighting linked events.

Figure 7.13: Object diagram of generic context to knowledge item linkage.

dividual phases in the meeting. In the minutes list, all versions of an individual QOC item are shown by choosing "Highlight Versions" in the context-menu as shown in Figure 7.15.



QOC Diagram

Q: Which Application concept?

	C: Roll-Out	C: Applic. Speed
O: Web	A-0	A-1

Revision Rationale

New Version of criteria: Roll-Out

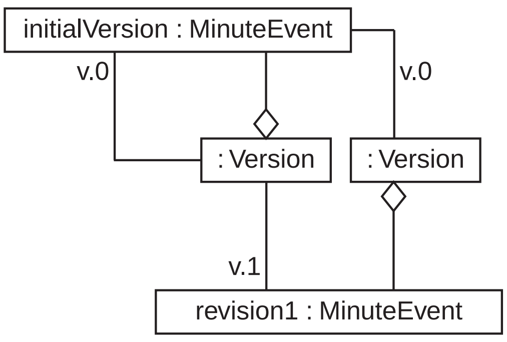
Deployment: User Training

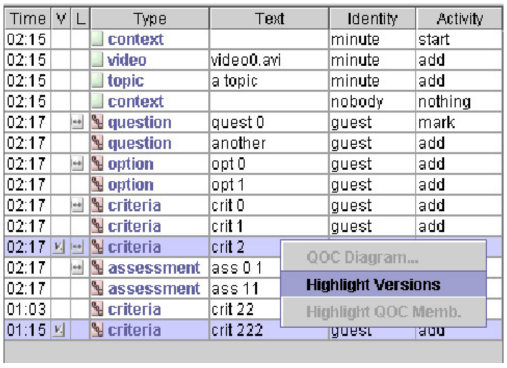
						Activity
						start
						add
						add
						nothing
						mark
						add
02:17			option	opt 0	guest	add
02:17			option	opt 1	guest	add
02:17			criteria	crit 0	guest	add

Figure 7.14: Add a revised version of an existing criteria, hence creating a new version.

By creating versions of QOC items, the resulting meeting minute history represents the actual history of the meeting. During brainstorming in a meeting, typically initial ideas and phrasings are replaced later with new versions. Storing those versions saves and visualizes the thinking process in the meeting and over a sequence of meetings.

Figure 7.15 shows the highlighting of linked versions for minute navigation. All subsequent and precedent versions of a selected knowledge item are selected and highlighted. Figure 7.12 shows how generic links are highlighted for a selected knowledge item or event in the meeting minutes. Figure 7.17 shows the highlighting of joined QOC items.

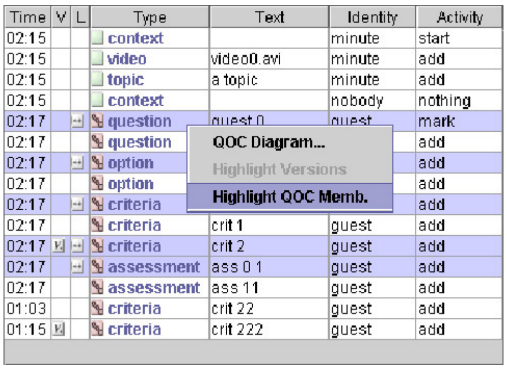




Time	V	L	Type	Text	Identity	Activity
02:15			context		minute	start
02:15			video	video0.avi	minute	add
02:15			topic	a topic	minute	add
02:15			context		nobody	nothing
02:17		↔	question	quest 0	guest	mark
02:17			question	another	guest	add
02:17		↔	option	opt 0	guest	add
02:17			option	opt 1	guest	add
02:17		↔	criteria	crit 0	guest	add
02:17			criteria	crit 1	guest	add
02:17	☑	↔	criteria	crit 2	[illegible]	[illegible]
02:17		↔	assessment	ass 0 1	[illegible]	[illegible]
02:17			assessment	ass 11	[illegible]	[illegible]
01:03			criteria	crit 22	[illegible]	[illegible]
01:15	☑		criteria	crit 222	guest	add

Figure 7.15: Highlight all versions of the selected criteria.

Figure 7.16: Object diagram for version linkage.



Time	V	L	Type	Text	Identity	Activity
02:15			context		minute	start
02:15			video	video0.avi	minute	add
02:15			topic	a topic	minute	add
02:15			context		nobody	nothing
02:17		↔	question	quest 0	guest	mark
02:17			question	[illegible]	[illegible]	add
02:17		↔	option	[illegible]	[illegible]	add
02:17			option	[illegible]	[illegible]	add
02:17		↔	criteria	[illegible]	[illegible]	add
02:17			criteria	crit 1	guest	add
02:17	☑	↔	criteria	crit 2	guest	add
02:17		↔	assessment	ass 0 1	guest	add
02:17			assessment	ass 11	guest	add
01:03			criteria	crit 22	guest	add
01:15	☑		criteria	crit 222	guest	add

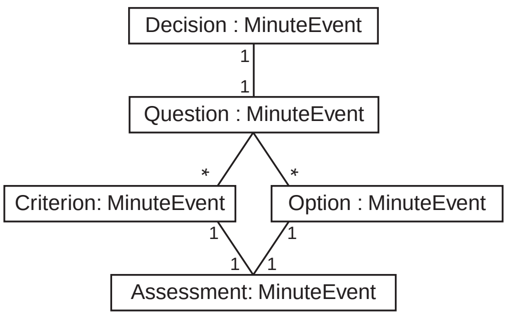


Figure 7.17: Meeting minute navigation by highlighting linked events.

Figure 7.18: Class diagram of QOC linkage.

The meeting minute is the central abstraction in the MEETINGGENERATOR. The meeting minute is mapped into a flat XML file. The entries in the meeting minute are structured time-based events, along with three types of links (generic, version, and QOC links). The meeting minutes refer to external data files (the video stream and snapshots) using URL references. A Data Type Definition (DTD) for the minutes is used to define the valid structure and entries as shown in the meeting minute model in Figure 7.19. The Simple API for XML (SAX) is used for XML parsing in the MEETINGGENERATOR. A Document Object Model (DOM) is used for meeting minute representation in memory (Figure 7.21).

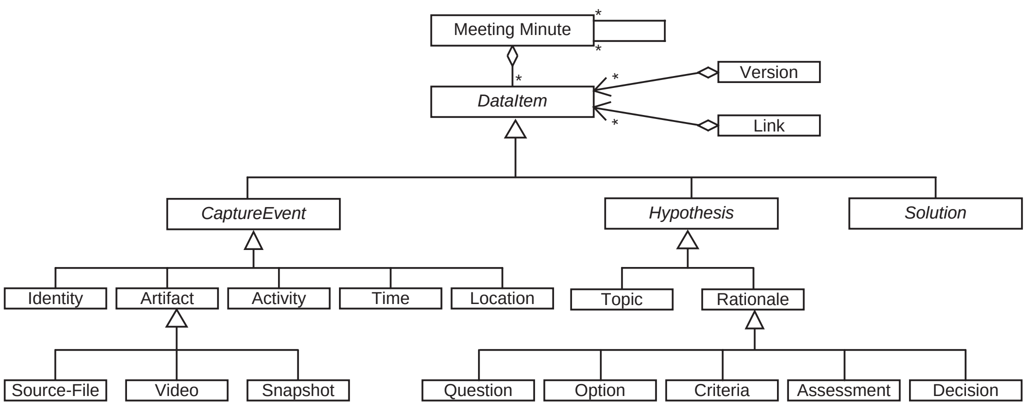


Figure 7.19: Taxonomy of data items and events stored in iBistro (UML class diagram).

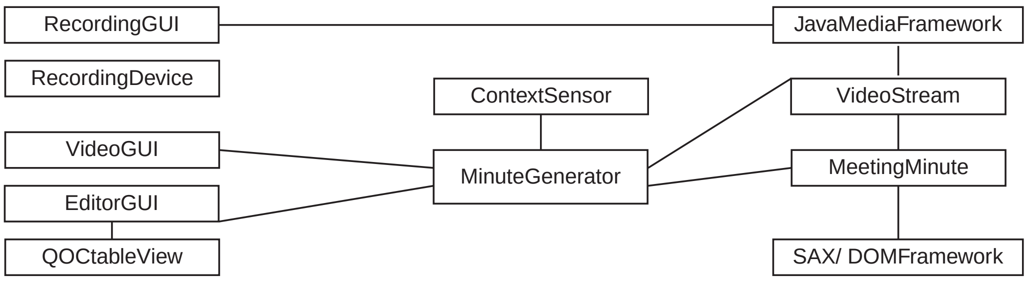


Figure 7.20: Objects and model of the MEETINGGENERATOR (UML class diagram).

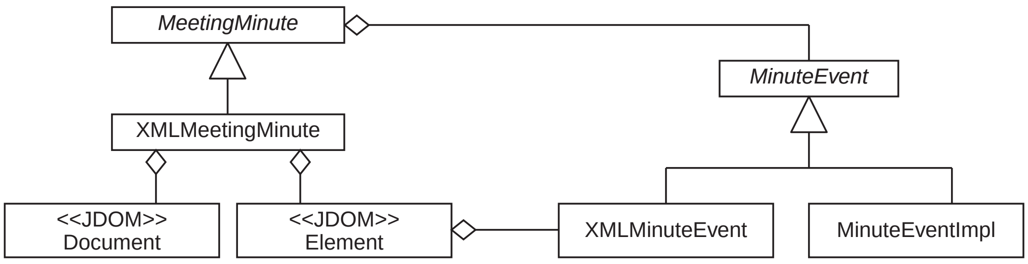


Figure 7.21: Model of the iBistro taxonomy using the JDOM API (UML class diagram).

7.1.3 CONCLUSION

During experimentation with the MEETINGGENERATOR tool we find that people using a PC or laptop computer during meetings disrupt the flow of a meeting. Hence, we investigated the possibilities of using roomware, such as SMART Boards^{TM}. SMART Boards are used to display information during a meeting and for instance for collaborative group UML editing. The SMART Boards are also used instead of a laptop or PC computer to be used by the meeting champion during the post-mortem process. The SMART Boards are big touch- sensitive displays to bring-up a computer's desktop screen which can be controlled manually without a mouse or even without a keyboard. Electronic pens provided with the SMART Boards are used to identify several different meeting participants by assigning each participant an individual pen color. We used the SMART Boards in status reports and meetings consecutively. The use of the large display increased the acceptance of electronic and software support during informal meetings compared to the initial use of a PC or laptop computer with the same software (for instance the MEETINGGENERATOR.)

Most of the reported limitations of the displays result from the specific type of SMART Boards used in the project, for instance:

- The SMART Boards in the MEETINGGENERATOR case study [Braun et al., 2001a] used a normal beamer with front projection. This typically led to partially shaded displays when a meeting participant stood between the beamer and the SMART Board.

- We used the iBistro server to run both the MEETINGGENERATOR and SMART Board tools and display drivers. The noise of the machines (server, beamer) disturbed the meetings. However, due to the limited length of the serial connection between the SMART Board and the controlling machine, the server had to stay in the same room.
  This issue is addresses by using a powerful but muted laptop computer.

- The SMART Board's touch screen is controlled by a serial connection when using a PC. This connection is rather slow and leads to delayed and back-logged recognition of the pen position, especially in continuous writing or drawing on the SMART Board.
  Meanwhile, this issue is addressed by the manufacturer. SMART Boards can now be connected to a PC using a variety of powerful connections, including Wireless LAN IEEE 802.11b WiFi and Ultra Serial Bus (USB).

In contrast to these limitations, the used meeting approach where post-mortem structuring takes place after the meeting was is largely accepted. This is not surprisingly, because it does not at all affect the meeting itself.

Beyond roomware and SMART Board usage, we identify the following lessons learned in iBistro: Context events are added to the MEETINGGENERATOR's meeting minutes list manually or by using the MEETINGGENERATOR as a Java API for context recording devices. Automatic context capture in the MEETINGGENERATOR proved to be not flexible enough due to the variety of potential context widgets. Much logic for context capture and processing would to be implemented in the MEETINGGENERATOR itself. As the use of third-party context widgets and tools is desirable re-use available technology and limit own work. However, third-party context capture APIs, for instance the Context Toolkit [Andind K. Dey, 2001], is difficult to integrate in the MEETINGGENERATOR.

The MEETINGGENERATOR encapsulated meeting minutes and post-mortem annotations in XML files. The meeting minute taxonomy and external artifacts, such as the audio/ video stream or whiteboard snapshots, are stored in flat files stored in a local filesystem.

This approach did not satisfy two requirements. First, the local filesystem does not work well in distributed settings, even though if the files are shared. In a shared filesystem, the entity shared is per file, which represents a whole meeting, instead of per event or knowledge item. Moreover, shared filesystems are too slow for Internet connections. Second, queries on flat files are slow and need to be implemented from scratch.

The lessons learned and issues identified during the first iteration led to an improved version of the MEETINGGENERATOR built on a specific software architecture as described in the next section.

7.2 ITERATION 2: THE SINGLE BLACKBOARD ARCHITECTURE

The MEETINGGENERATOR used in the first iteration was a stand-alone solution to capture and annotate a single-site meeting. The distributed software architecture provides a technical infrastructure for iBistro. (Discussed in Chapter 6.) Using this infrastructure, MEETINGGENERATOR becomes a collaborative tool to be used from several project sites.

The software architecture for iBistro further addresses the issues identified during experimentation (see Iteration 1, Section 7.1.3) with the MEETINGGENERATOR, specifically:

- The *integration of remote context-aware and roomware devices* addresses the usability and context capturing issues.
- An *active repository* enables automatic context and event processing and enables intelligent suggestions made by the system.

- A *distributed knowledge repository* allows for knowledge cross-linking between several sites and knowledge reuse independent of the user's location.
- A *common programming architecture and framework* enables easier development of knowledge sources and experimentation with the system.

To meet with the experimental plan at Singapore, we split the architecture into two versions:

The first version, the *single blackboard architecture* approach was chosen to quickly implement an initial version that consecutively would be refined during testing and evaluation. The single blackboard practically has all important functional features to deal with distributed settings in general, such as random access from remote sites over the internet, a common knowledge repository, and so on.

The experiences with the single blackboard architecture are then addressed in the second version, the *distributed concurrent blackboard architecture* (DCBA). In this version, we first focused on problems identified in everyday usage of the iBistro system, for instance performance, ease of use and extendability, and bugs. We then improved the approach by extending the iBistro capabilities for transparent access of iBistro resources from remote sites and by making it a net-centric system with all resources and knowledge items stored in a distributed manner on several sites.

7.2.1 SCOPE

The single blackboard architecture implements a framework for iBistro consisting of one central blackboard with control and strategy components, several remote knowledge sources and remote knowledge views. The blackboard serves as one single central repository with remote components connected via network connections based on JINI/ Java RMI services. This setting provides easy collaboration, as there is no need for knowledge replication or synchronization for several concurrent instances. The central blackboard is able to coordinate all queries from remote components.

7.2.2 APPROACH

Figure 7.22 shows the single blackboard installation in Munich and several local and remote components of the system. All data items are forwarded to the single blackboard server in Munich. Remote knowledge sources will work on the data and knowledge items using the rather slow internet connection between Munich and Singapore.

The single blackboard architecture implements the basic iBistro architecture based on the standard blackboard model. The single blackboard architecture is

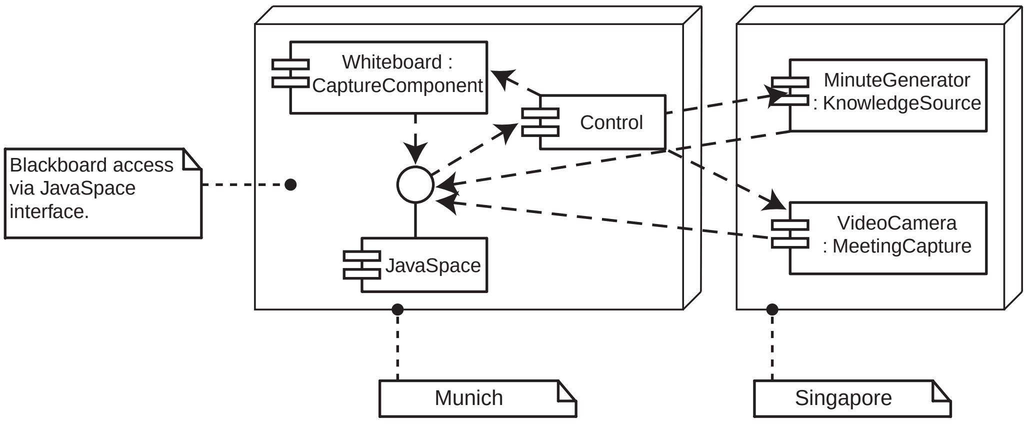


Figure 7.22: Setup of the first experiment between Singapore and Munich (UML component diagram).

implemented based on the JavaSpaces framework [Sun Microsystems, 2003]. In the case study, two teams collaborate in the same project by accessing a central Munich-based iBistro server. Meeting room devices (capture components) and knowledge sources connect to the central repository via LAN or WAN connections. Remote components have to use a narrow and slow Internet connection for all data transferred.

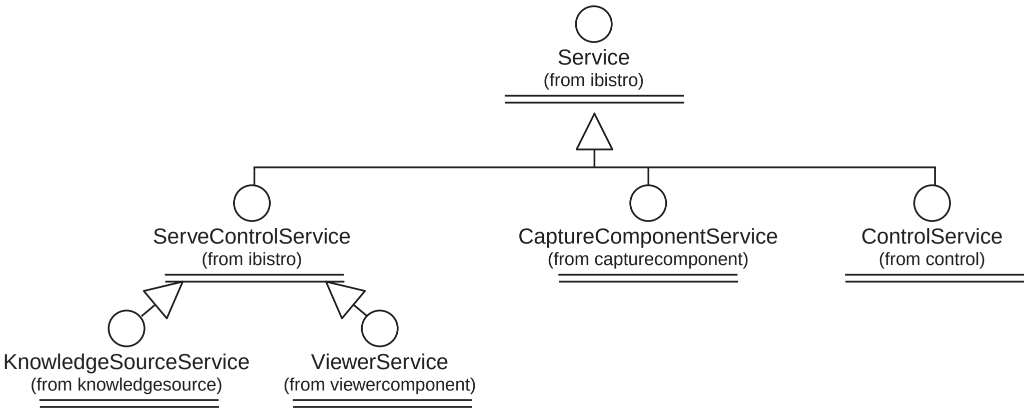


Figure 7.23: Hierarchy of iBistro services used in the single blackboard architecture (UML class diagram).

The remote components in the iBistro architecture are based on several iBistro services as shown in Figure 7.23. iBistro services extend the Java Remote Interface to enable remote invocation. Each interface provides methods to register and unregister an iBistro service in the iBistro architecture. There are three services derived from the basic iBistro remote service (`Service`). iBistro components scheduled by the control component are based on the `ServeControlService`. The two scheduled services in iBistro are the `KnowledgeSourceService` for knowledge sources and the `ViewerService` for knowledge views and browser components. The `CaptureComponentService` provides connectivity for non-scheduled components, especially context-capturing components to interface sensors. The `ControlService` finally connects the scheduling instance, the Control component.

7.2.3 LESSONS LEARNED AND NEXT STEPS

The single blackboard architecture implemented a basic framework for further testing in the distributed scenario. While the single blackboard system did not yet meet all the desirable features for iBistro, the following goals were achieved:

- The implementation of a basic framework for further evaluation and improvement. The framework allows the integration of knowledge sources and capture components, as well as the MEETINGGENERATOR.
- A distributed, while central, active knowledge repository.
- A test harness replaces context-sensitive devices that were not available to the project (for instance active location tracking is replaced by a small PC tool to allow passive user tracking).
- Knowledge sources and capture components for whiteboard capturing were implemented for the case study.

Two major design drawbacks were identified during the distributed experiment. First, the lack of persistent storage. The knowledge and content, except artifacts, were stored in Java objects only in the single blackboard architecture. Whenever the application was stopped and restarted, the content of the Java memory was lost. The files storing the artifacts, however, stayed available in the file system, but without any attached information in the taxonomy. Second, the lack of powerful queries on the knowledge space. Even simple queries are very time consuming in the knowledge space. Complex queries, such as “list all programs written by User A before January in Singapore”, were not possible.

Most of the other problems encountered were primarily related to performance[2] and system stability in distributed settings. Performance becomes specifically critical when many local context-events are promoted to the central knowledge space. All these issues were consecutively addressed in the DCBA architecture.

[2]A Benchmark test of the single blackboard architecture vs. the DCBA is discussed and illustrated in Section 7.3 in Table 7.3.

7.3 ITERATION 3: THE DISTRIBUTED CONCURRENT BLACKBOARD ARCHITECTURE

7.3.1 SCOPE

The distributed concurrent blackboard architecture (DCBA) is designed and built to address the specific requirements identified during the project communication study and the case study between Singapore and Munich. The DCBA architecture realizes a distributed knowledge repository to support balanced teams by providing each team with a local blackboard [Nikitsch, 2003].

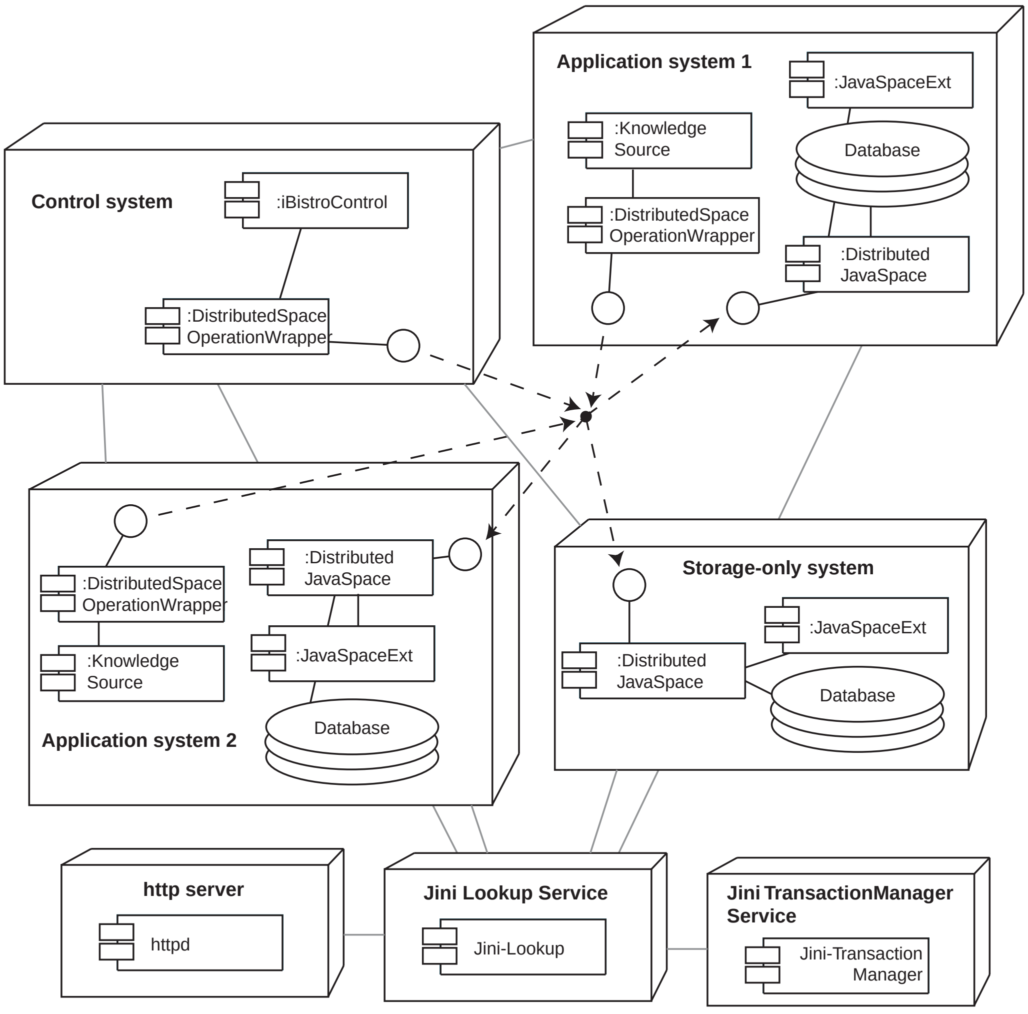


Figure 7.24: DCBA deployment diagram.

7.3.2 APPROACH

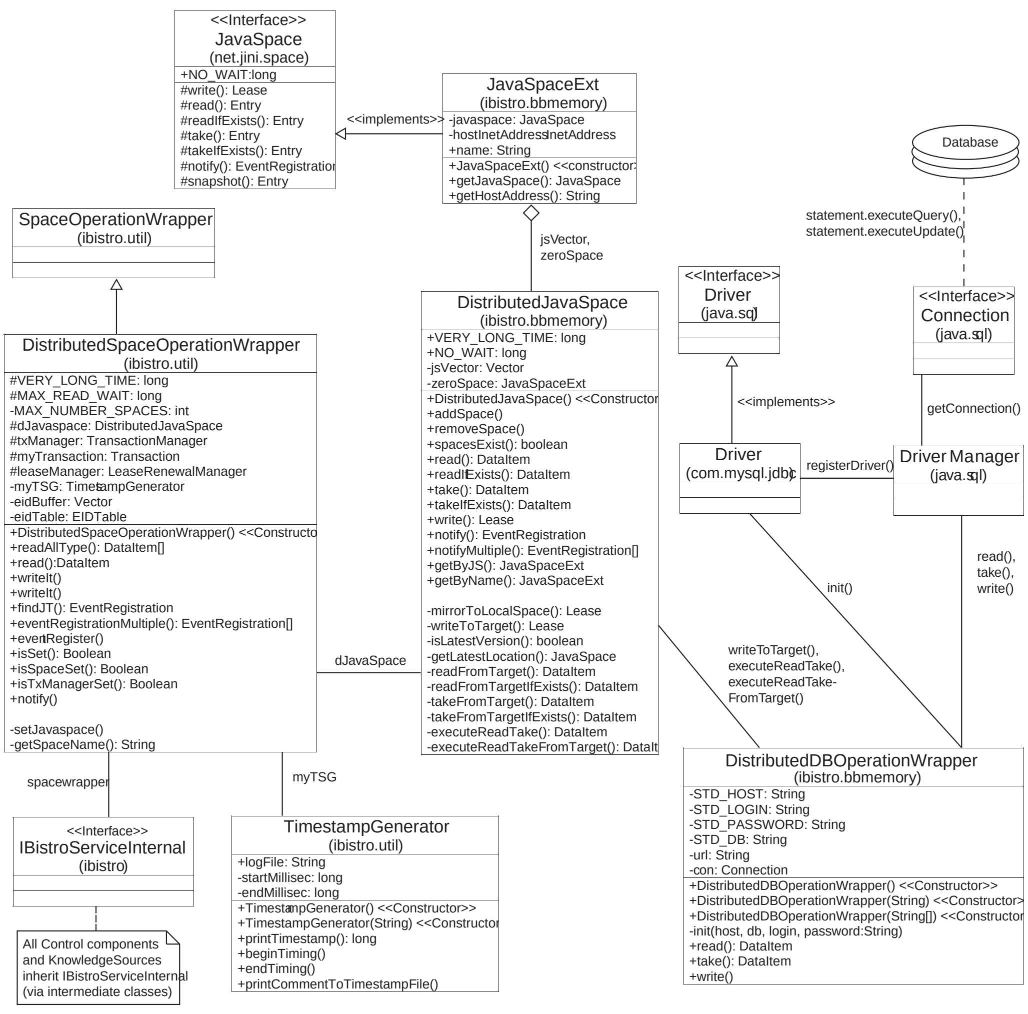
The local blackboard, while enabling high performance access for on-site work and queries, is still distributed in that it is accessible from remote sites seamlessly as well. The DCBA creates a local environment for technical components, such as knowledge sources or context-capture devices, as well as local team members. All replication and synchronization of data and knowledge items is handled within the framework.

The second important point is that as the DCBA framework is built step-by-step through testing and evaluation. It is much more stable and debugged compared to the first version used in the single blackboard architecture. During continuous testing, however, one more problem had been identified in the DCBA: for simplicity of knowledge source scheduling, the first release of the DCBA had, differing from the design (see Figure 6.2 in the previous chapter) only one central control component. While this allows for easy global knowledge source scheduling, the main drawback is that all notification has to be sent to the singular control component. In case of event notification in an iBistro room with many context-aware devices, for instance, this results in tremendous network traffic. This issue is addressed with cascaded control. In the cascaded control, any event is first propagated through the local instance of the control component. Only if the local control component could not handle the event, it is forwarded to global control (by simply following the inheritance hierarchy of control components).

The DCBA stores data items in a relational database system (mySQL [MySQL, 2003]) for persistent storage. The virtual tier (see Figure 6.3) serves as the global infrastructure and middleware and acts like a caching device between the components and the persistent storage, hence hiding the underlying storage architecture.

7.3.3 LESSONS LEARNED

Test scenarios with globally distributed teams who are developing software collaboratively are expensive to set up. To further test the architectural framework, the scenarios are simulated by automatically generating contextual events. Slow network connections are simulated by using wireless network cards which allow to reduce the transfer speed manually. The iBistro test harness allows to simulate many of the events which might occur in a distributed project setting. The iBistro control center allows to start, stop, and manually add or remove components and hence simulate additional sites, or crashing of components. Even sites that are temporarily unavailable, for instance due to network connections lost, can be simulated. Moreover, the test harness can be used to add testing how the framework would work with components which are not yet actually imple-



<<Interface>> JavaSpace (net.jini.space)
+NO_WAIT:long
#write(): Lease
#read(): Entry
#readIfExists(): Entry
#take(): Entry
#takeIfExists(): Entry
#notify(): EventRegistration
#snapshot(): Entry
<<implements>>
JavaSpaceExt (ibistro.bbmemory)
-javaspace: JavaSpace
-hostInetAddress: InetAddress
+name: String
+JavaSpaceExt() <<constructor>>
+getJavaSpace(): JavaSpace
+getHostAddress(): String
Database
statement.executeQuery(), statement.executeUpdate()
SpaceOperationWrapper (ibistro.util)
jsVector, zeroSpace
<<Interface>> Driver (java.sql)
<<Interface>> Connection (java.sql)
DistributedJavaSpace (ibistro.bbmemory)
+VERY_LONG_TIME: long
+NO_WAIT: long
-jsVector: Vector
-zeroSpace: JavaSpaceExt
+DistributedJavaSpace() <<Constructor>>
+addSpace()
+removeSpace()
+spacesExist(): boolean
+read(): DataItem
+readIfExists(): DataItem
+take(): DataItem
+takeIfExists(): DataItem
+write(): Lease
+notify(): EventRegistration
+notifyMultiple(): EventRegistration[]
+getByJS(): JavaSpaceExt
+getByName(): JavaSpaceExt
-mirrorToLocalSpace(): Lease
-writeToTarget(): Lease
-isLatestVersion(): boolean
-getLatestLocation(): JavaSpace
-readFromTarget(): DataItem
-readFromTargetIfExists(): DataItem
-takeFromTarget(): DataItem
-takeFromTargetIfExists(): DataItem
-executeReadTake(): DataItem
-executeReadTakeFromTarget(): DataItem
DistributedSpaceOperationWrapper (ibistro.util)
#VERY_LONG_TIME: long
#MAX_READ_WAIT: long
-MAX_NUMBER_SPACES: int
#dJavaspace: DistributedJavaSpace
#txManager: TransactionManager
#myTransaction: Transaction
#leaseManager: LeaseRenewalManager
-myTSG: TimestampGenerator
-eidBuffer: Vector
-eidTable: EIDTable
+DistributedSpaceOperationWrapper() <<Constructor>>
+readAllType(): DataItem[]
+read():DataItem
+writeIt()
+writeIt()
+findJT(): EventRegistration
+eventRegistrationMultiple(): EventRegistration[]
+eventRegister()
+isSet(): Boolean
+isSpaceSet(): Boolean
+isTxManagerSet(): Boolean
+notify()
-setJavaspace()
-getSpaceName(): String
<<implements>>
getConnection()
Driver (com.mysql.jdbc)
registerDriver()
Driver Manager (java.sql)
read(), take(), write()
init()
writeToTarget(), executeReadTake(), executeReadTake-FromTarget()
dJavaSpace
spacewrapper
myTSG
DistributedDBOperationWrapper (ibistro.bbmemory)
-STD_HOST: String
-STD_LOGIN: String
-STD_PASSWORD: String
-STD_DB: String
-url: String
-con: Connection
+DistributedDBOperationWrapper() <<Constructor>>
+DistributedDBOperationWrapper(String) <<Constructor>>
+DistributedDBOperationWrapper(String[]) <<Constructor>>
-init(host, db, login, password:String)
+read(): DataItem
+take(): DataItem
+write()
<<Interface>> IBistroServiceInternal (ibistro)
TimestampGenerator (ibistro.util)
+logFile: String
-startMillisec: long
-endMillisec: long
+TimestampGenerator() <<Constructor>>
+TimestampGenerator(String) <<Constructor>>
+printTimestamp(): long
+beginTiming()
+endTiming()
+printCommentToTimestampFile()
All Control components and KnowledgeSources inherit IBistroServiceInternal (via intermediate classes)

Figure 7.25: The final implementation of the distributed blackboard architecture.

mented (like we did with automatic location tracking of team members during the first iteration).

Tables 7.2 and 7.3 show performance benchmarks of the distributed blackboard architecture [Nikitsch, 2003]. The benchmarks were made based on the system and hardware configuration shown in Table 7.1. For the following comparison it is important to note that the DCBA architecture also benefits from hardware performance. The table shows that the system Munich is much more powerful than the laptop computer Singapore.

Table 7.2 compares the speed of a cached blackboard access to a request that needs full replication from the database for the two installations at Munich and Singapore. The comparison shows the improvements in performance and especially in response time for cached queries. The comparison also shows that the

Munich:	Pentium 4, 1.6 GHz, 256 MB RAM, 30 GB hard disk, Windows XP Professional, Java 1.4.1 SE, Jini 1.2.1, MySQL 3.23
Singapore:	Pentium 3, 500 MHz, 192 MB RAM, 30 GB hard disk, Windows 2000 Professional, Java 1.4.1 SE, Jini 1.2.1, MySQL 3.23
LAN:	100 MBit/s Ethernet LAN
WLAN:	11 MBit/s WLAN (IEEE 802.11), both locally as well as in the intranet of the National University of Singapore

Table 7.1: System and hardware configuration for the Benchmark.

overall DCBA performance benefits largely from hardware performance.

Location	Object size (KBytes)	get() from cache	put() to cache	DCBA restore from db	DCBA restore from db
Singapore	< 10	0.23 s	0.30 s	0.27 s	0.90 s
	700	2.00 s	6.60 s	2.50 s	8.50 s
Munich	< 10	0.15 s	0.17 s	0.17 s	0.37 s
	700	0.55 s	0.88 s	1.10 s	2.20 s

Table 7.2: Benchmark: Cached objects versus objects restored from database.

Table 7.3 compares the performance of the local system in Singapore with remote access from Singapore to Munich. The table shows that remote access of cached objects is nearly as fast as local access in Singapore even though object IDs have to be checked whether replication is needed. For very large amounts of data items (one million objects), a $\approx 21\%$ increase in access time is recorded.

	# of data items	get()	connect	connect +get()	put()	connect	connect +put()
Singapore	100	4.66 ms	490 ms	495 ms	5.61 ms	520 ms	526 ms
	10,000	140 ms	490 ms	630 ms	5.51 ms	510 ms	516 ms
	1,000,000	72.9 s	510 ms	73.4 s	6.57 ms	520 ms	527 ms
Munich	100	2.10 ms	200 ms	202 ms	2.10 ms	210 ms	212 ms
	10,000	60.9 ms	200 ms	261 ms	2.24 ms	200 ms	402 ms
	1,000,000	16.0 ms	200 ms	16.2 ms	6.25 ms	210 ms	835 ms
Indexed	1,000,000	5.92 s	200 ms	6.12 s	375 ms	200 ms	575 ms
remote access	100	4.76 ms	510 ms	515 ms	5.84 ms	500 ms	506 ms
	10,000	111 ms	490 ms	601 ms	5.11 ms	510 ms	515 ms
	1,000,000	92.9 ms	500 ms	93.4 ms	5.10 ms	510 ms	515 ms

Table 7.3: Benchmark: Local versus remote DCBA access time.

Overall, the DCBA displayed much faster reactions to user requests and queries than the single blackboard architecture. Due to cached saving, the DCBA also allows knowledge sources to write large amounts of data, for instance video streams or captured whiteboard images, or large numbers of items, for instance sensor data, fast and in quick sequences.

7.4 DISCUSSION

iBistro has been developed iteratively in three stages through experimentation in local and distributed settings. During the case studies, we had a strong focus on an architecture that allows us to capture as much as possible of informal meetings and provides that information to distributed stakeholders. As the subjects of our investigation were primarily students and research assistants, we encountered low resistance from users, if at all. However, especially during video and audio capturing of meetings and automatic tracking of users and user activity, we were faced with privacy concerns. iBistro does not yet support any means of protecting the privacy of users, such as allowing only certified users to see specific entries. As iBistro is meant to be a research system to investigate the basic principles of cooperation based on knowledge from informal meetings in global distributed software development, we also did not implement any encryption or protection beyond the capabilities, provided by the underlying operating systems.

CHAPTER 8

DISCUSSION

Die höchste Philosophie des Naturforschers besteht eben darin, eine unvollendete Weltanschauung zu ertragen und einer scheinbar abgeschlossenen, aber unzureichenden vorzuziehen.
Die Mechanik in ihrer Entwicklung, [ERNST MACH]

The only thing that stays the same is change.
Change, [MELISSA ETHERIDGE]

From a functional perspective, iBistro supports globally distributed software development. This includes the early and non-technical phases in software development, such as writing proposals and requirements elicitation. iBistro supports globally distributed software development teams by providing organizational knowledge which is built-up by a network of many linked and connected instances of group memories, for instance recorded and annotated meetings. This network of knowledge stores the information, issues, argumentation, and artifacts of several projects.

iBistro focuses on distributed teams as opposed to individuals by providing a common infrastructure to make the distribution as transparent as possible. The distributed concurrent blackboard architecture interlinks pieces of knowledge assembled by global teams and enables both distributed usage and hindsight usage by providing the necessary contextual information, such as rationale or argumentation. An ontology allows for information to knowledge transition and knowledge crossover. Finally, iBistro supports informal meetings and brainstorming sessions as source of information and knowledge in a distributed context.

On the technical side, iBistro provides a transparent infrastructure for knowledge acquisition and search for solutions in a distributed concurrent blackboard architecture (DCBA). The DCBA is an extension of the standard blackboard architecture. The DCBA enables the concurrency of components and adds a distributed shared repository. Moreover, the DCBA realizes persistent data storage in a relational database management system. The database also enables SQL to be used as query language for all components in the system. The iBistro infrastructure is encapsulated in a Java framework with base implementations for all suggested components of the system, such as knowledge sources, capture components, views, and the blackboard itself.

The research prototypes developed for evaluation of the system provide specific knowledge sources for audio and video capturing, meeting minute editing, viewing, and testing. In related efforts and during a one semester software development course, support for roomware has been evaluated using SMART Boards^{TM}. Building a group memory that enables the integration of various communication media is now possible using the suggested common framework for future developments.

8.1 OBSTACLES

iBistro has been used as a research system for data collection and experimentation in distributed projects at universities during our experiment between the National University of Singapore and TU-München. The subjects working with the system so far were either students or researchers.

The envisioned user group for iBistro are project members in larger projecbased organizations. In the following, we identify and discuss issues in using iBistro related to the technical and organizational aspects of the system.

8.1.1 STANDARDIZATION OF INFRASTRUCTURES

iBistro suggests a standardized infrastructure which is based on JINI services in the implementation. This architecture serves as a middleware and is based on existing technology. The infrastructure connects a variety of devices, including 'ubiquitous' devices in an active room. However, to include recent and upcoming technology, knowledge sources might serve as a wrapper for those devices or communication media to interface the iBistro system. This could be useful for e.g. Bluetooth devices.

To be usable in a broader context, the iBistro infrastructure and software architecture is to be made available throughout an organization as a whole.

CSCW systems in general have to be accepted by all members of the user group. Different users have differing preferences on tools, they have different expectations, different value-oriented attitudes, and a different background (for instance company, professional context, or education). Moreover, groupware systems have to consider various actual roles in a workgroup.

8.1.2 SOCIAL, ORGANIZATIONAL, & MANAGEMENT IMPEDIMENTS

iBistro aims at capturing as much information and knowledge within an organization as possible. In iBistro, a variety of knowledge sources are using many different automated and semi-automated approaches to capture information in informal meetings and from other sources. As such knowledge sources are continuously and dynamically added to the system, users will likely lose the perception of what information is actually captured and may get the feeling of being monitored and restricted in their privacy. This can result in resistance from users, clients, and from the works council, if applicable.

Social, security, and privacy issues have to be addressed prior to rolling the system out to a professional environment. Privacy issues are currently under investigation in the GSE research group at the Chair for Applied Software Engineering with two specific focuses: First, user models and user profiles will be developed to allow users to view and alter the way any information related to their person is used. (This is done in the “Smart Aquarium” sub-project shown in Table 8.1.) Second, the information should be encrypted. A public key method could be used to encrypt any event (or data item) using a freely available public key. Only components (knowledge sources and views) authorized by the user receive the suitable private key to decrypt and therefore use or propagate the knowledge.

Beyond the social, security, and privacy issues which may result in resistance from users, resistance from management is expected due to the introduction of new and risky to use technology. Given sufficient experience with the system and approaches in an academic environment, the system promises to address upcoming issues in distributed projects. If the current trend to reduce development cost and time to market by offshore software development continues, the system by addressing many of the key issues would experience low resistance from management.

On the organizational side, however, specific users would have to be assigned to fulfill specific tasks in the iBistro system, such as the meeting champion and so on. This potentially affects organizational culture and the human resources department.

8.2 RELATED AND ONGOING RESEARCH

The Global Software Engineering (GSE) group an the chair for applied software engineering at Technische Universität München works towards understanding the major collaboration issues in distributed and global software development. The group develops integrated tools and guidance for supporting the collaboration of globally distributed teams and evaluates these methods and tools in an empirical environment.

Effective solutions will include a combination of technical, social, methodological factors. Hence, the GSE group adopts an empirical approach using around a distributed software engineering project course as an experimental environment.

Sub-Project	Infra-structure	Col-laboration	Knowledge Management	Artifact Management	App-liance	HCI
iBistro	✓	✓	✓		✓	
ABX				✓		
ADReaM					✓	✓
REQuest			✓		✓	
Smart Aquarium				✓	✓	✓

Table 8.1: Related and ongoing research in the GSE group.

8.2.1 ABX: GROUP AWARENESS

ABX focuses on improving awareness across sites and project participants. ABX collects events generated by development and communication tools and then uses user-specified interests and system rules to determine which events may be interesting for a specific user.

8.2.2 ADREAM: DISTRIBUTED MODELING

Augmented Distributed Real-time Modeling (ADReaM) focuses on providing an Augmented Reality environment for supporting the real time collaboration of small groups of developers over UML and QOC models. Developers use augmented reality to manipulate and annotate a virtual model on a table top and to collaborate from different sites.

8.2.3 REQUEST: RATIONALE USE IN DISTRIBUTED SOFTWARE ENGINEERING

Rationale-based Use Case Specification (REQuest) focuses on capturing and leveraging off explicit decision making knowledge during requirements specification. Participants write use cases using a web-based collaborative tool, and model questions, issues, challenges, and justifications using QOC matrices.

8.2.4 SMART AQUARIUM: USER MODELLING

The “smart aquarium” project focuses on the Human-Computer-Interaction in a context-aware environment. With help of an rule-based framework, the end users are given the control over such an environment and can build customized context-aware applications without programming skills.

8.3 OUTLOOK AND FUTURE WORK

8.3.1 DOMAIN EXPERT KNOWLEDGE AND INTELLIGENT SUPPORT

With techniques from intelligent tutoring systems (ITS) and computer supported collaborative learning (CSCL) systems we can get a learning environment with many facilities for supporting work and learning processes: the means for this encompassing support are the use of student models and the explicit representation of expert knowledge in the problem domain (in our case software engineering). Student models enable us to address one of the problems described in the introduction, that is the inability to find stakeholders quickly. With a combination of self-assessment from the student´s side and diagnosis of the student´s problem solving behavior we can get a representation of the student´s capabilities, expertise and weaknesses, that is a student model. This can be used to help the students finding stakeholders and experts in certain areas much more quickly, just by requesting help from the system to get a recommendation which person should be contacted. For example a student in the software engineering course, whose task is the implementation of a subsystem, runs into problems with the design the team planned. At that moment he needs the help of a design expert in the team. Based on self-assessment of the students and on diagnosis of the previous work, iBistro could recommend a team member meeting the criteria the other student asks for. The process for finding specific stakeholders or roles is very similar to that in Opportunistic Group Formation [Ikeda et al., 1997], a well-known procedure in the field of computer supported collaborative learning. With the explicit representation of expert domain knowledge (here with the topic of software engineering, like process models, rationale, design, and its refactoring) the learning environment iBistro may also provide intelligent support on its own, if a human expert is not available (due to asynchronous work or different time zones). The fundament of that expert knowledge is the definition of an ontology of the domain, which defines all the important terms and relations of the expert domain. We plan to adapt an authoring tool for the definition of concept maps and ontologies. For the user-interface of the artificial domain expert we propose the technique of synthetic interviews and synthetic agents [Johnson, 1999], that provide an artificial anthropomorphic partner for the human group members.

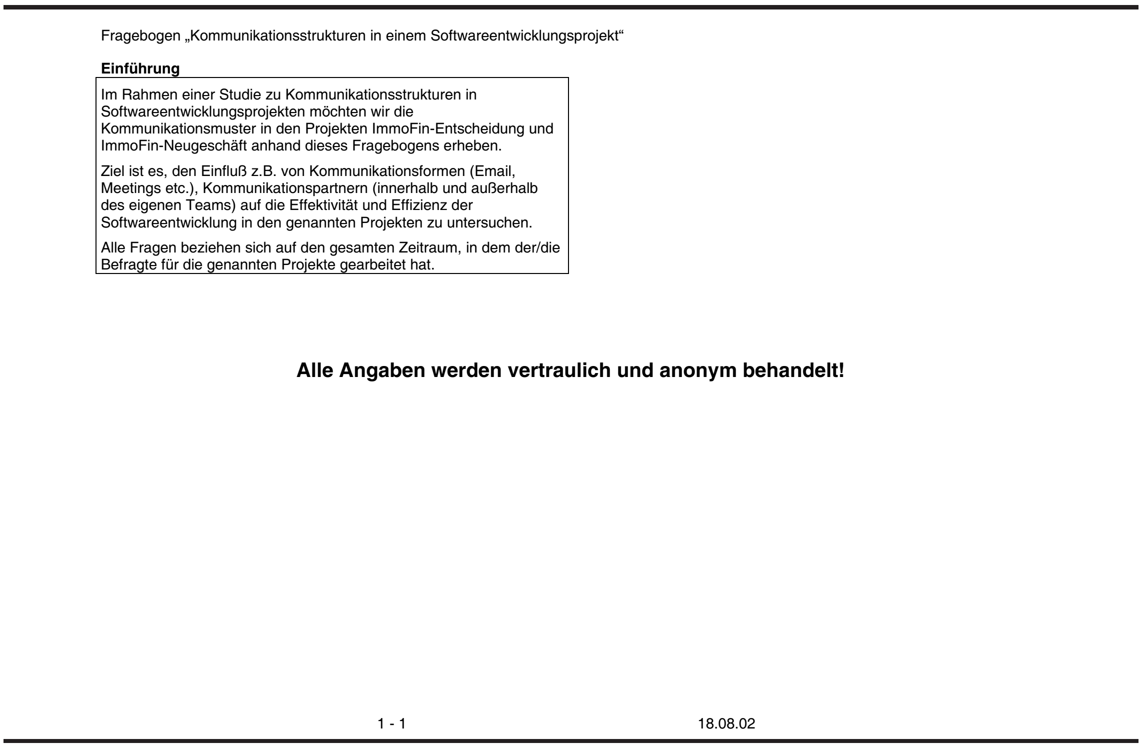
8.3.2 FUTURE STUDIES

Our major concern regarding future studies are extendability to larger/ different populations and repeatability. One could plan to extend the survey to a group of students at Technische Universität München (TUM). The main interest would be to assess the comparability of professionals (consultants) and master students. This idea is motivated by the fact that samples for early tool evaluation and empirical software engineering are found much easier at universities than in industry. Moreover, universities provide a suitable environment for prototype and tool development in many aspects. However, one typical allegation, especially from industry, is that students are not comparable to professionals for several reasons (not working exclusively for one project, no payment, less experienced, loose organizational ties, …). Our goal for a follow-up survey is to identify and assess the differences between professional consultants (as investigated in this study) and students. If feasible, we aim at defining an environment that allows for professional-style software development at universities (for instance software development internships with 'real' client interaction).

APPENDIX A

RESEARCH MATERIAL

A.1 INTERVIEW GUIDE



Fragebogen „Kommunikationsstrukturen in einem Softwareentwicklungsprojekt"

Einführung

Im Rahmen einer Studie zu Kommunikationsstrukturen in Softwareentwicklungsprojekten möchten wir die Kommunikationsmuster in den Projekten ImmoFin-Entscheidung und ImmoFin-Neugeschäft anhand dieses Fragebogens erheben.

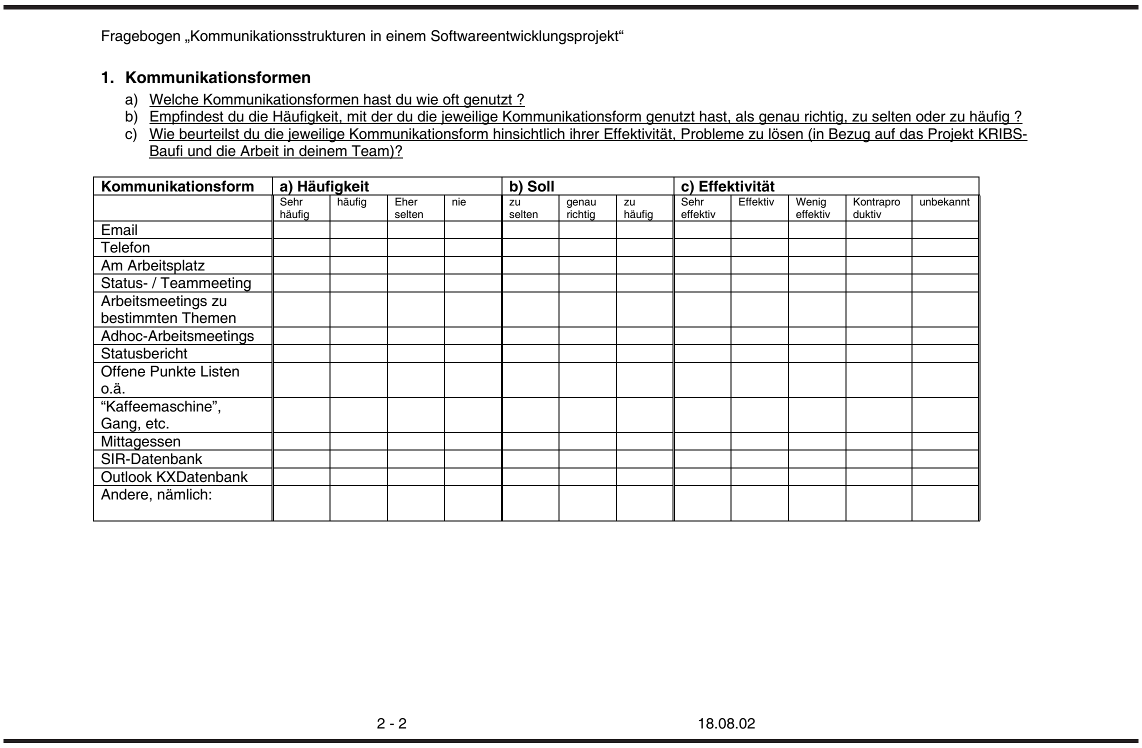
Ziel ist es, den Einfluß z.B. von Kommunikationsformen (Email, Meetings etc.), Kommunikationspartnern (innerhalb und außerhalb des eigenen Teams) auf die Effektivität und Effizienz der Softwareentwicklung in den genannten Projekten zu untersuchen.

Alle Fragen beziehen sich auf den gesamten Zeitraum, in dem der/die Befragte für die genannten Projekte gearbeitet hat.

Alle Angaben werden vertraulich und anonym behandelt!

1 - 1 18.08.02

Figure A.1: Interview guide page 1



Fragebogen „Kommunikationsstrukturen in einem Softwareentwicklungsprojekt"

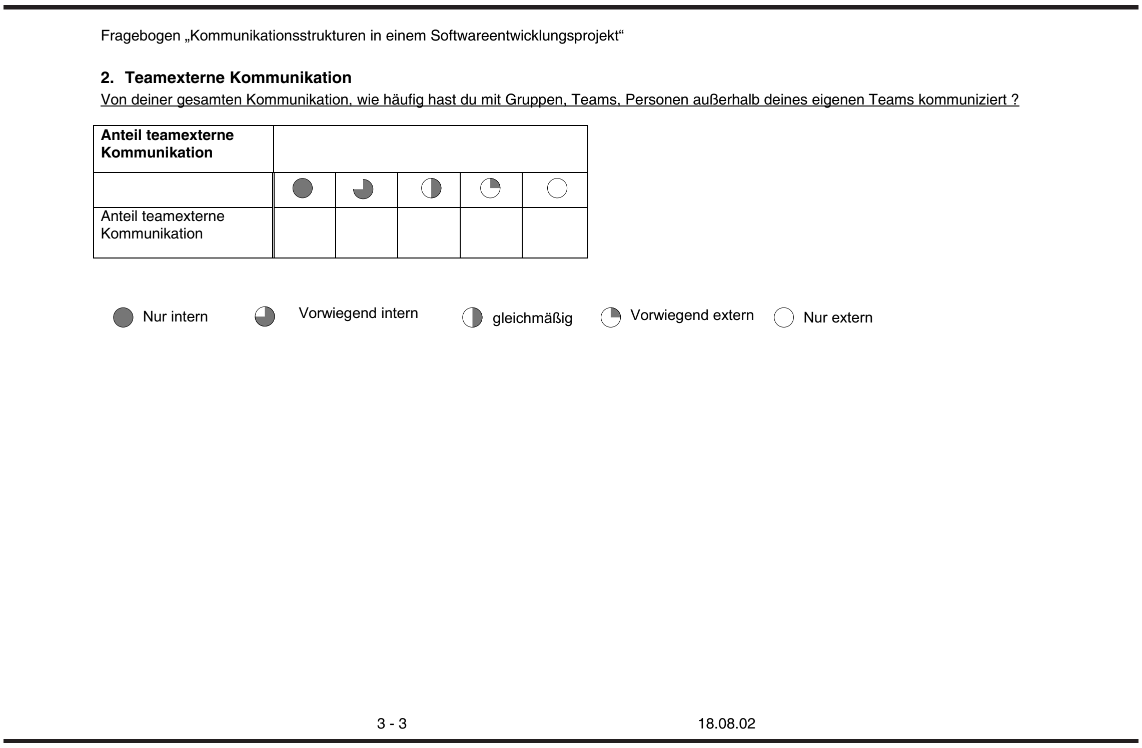
1. Kommunikationsformen

a) Welche Kommunikationsformen hast du wie oft genutzt ?
b) Empfindest du die Häufigkeit, mit der du die jeweilige Kommunikationsform genutzt hast, als genau richtig, zu selten oder zu häufig ?
c) Wie beurteilst du die jeweilige Kommunikationsform hinsichtlich ihrer Effektivität, Probleme zu lösen (in Bezug auf das Projekt KRIBS-Baufi und die Arbeit in deinem Team)?

Kommunikationsform	**a) Häufigkeit**				**b) Soll**			**c) Effektivität**				
	Sehr häufig	häufig	Eher selten	nie	zu selten	genau richtig	zu häufig	Sehr effektiv	Effektiv	Wenig effektiv	Kontraproduktiv	unbekannt
Email												
Telefon												
Am Arbeitsplatz												
Status- / Teammeeting												
Arbeitsmeetings zu bestimmten Themen												
Adhoc-Arbeitsmeetings												
Statusbericht												
Offene Punkte Listen o.ä.												
"Kaffeemaschine", Gang, etc.												
Mittagessen												
SIR-Datenbank												
Outlook KXDatenbank												
Andere, nämlich:												

2 - 2 18.08.02

Figure A.2: Interview guide page 2



Fragebogen „Kommunikationsstrukturen in einem Softwareentwicklungsprojekt“

2. Teamexterne Kommunikation

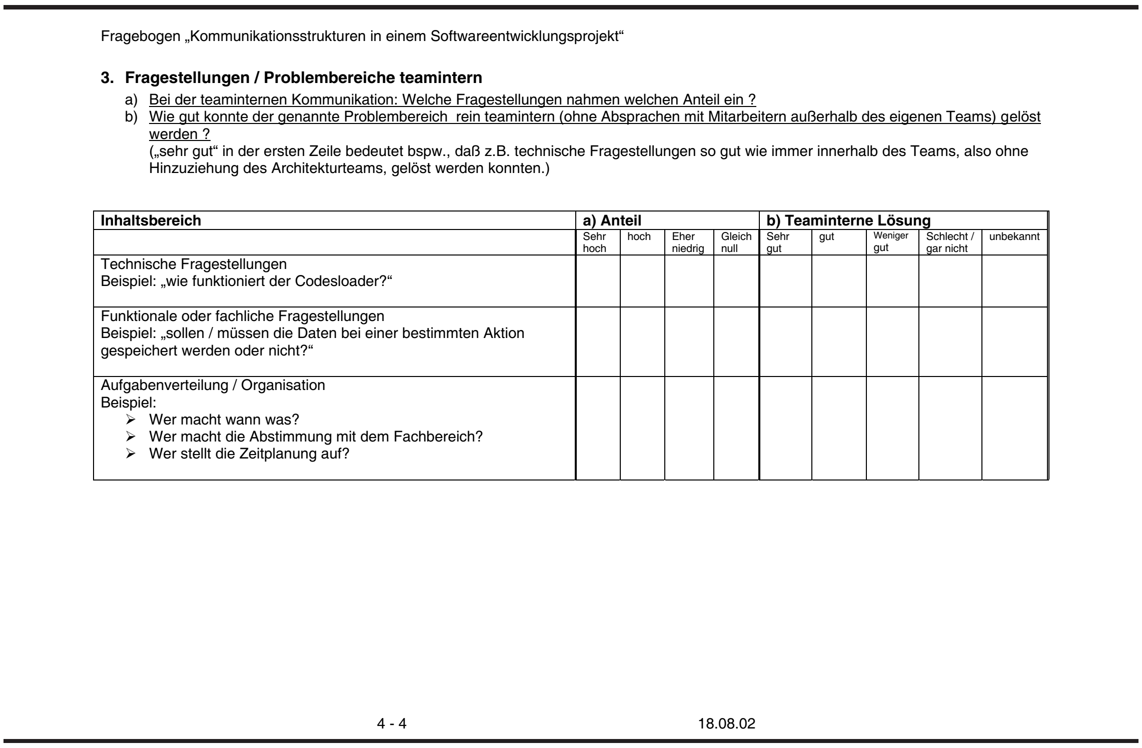
Von deiner gesamten Kommunikation, wie häufig hast du mit Gruppen, Teams, Personen außerhalb deines eigenen Teams kommuniziert ?

Anteil teamexterne Kommunikation					
	●	◕	◑	◔	○
Anteil teamexterne Kommunikation					

● Nur intern ◕ Vorwiegend intern ◑ gleichmäßig ◔ Vorwiegend extern ○ Nur extern

3 - 3 18.08.02

Figure A.3: Interview guide page 3



Fragebogen „Kommunikationsstrukturen in einem Softwareentwicklungsprojekt“

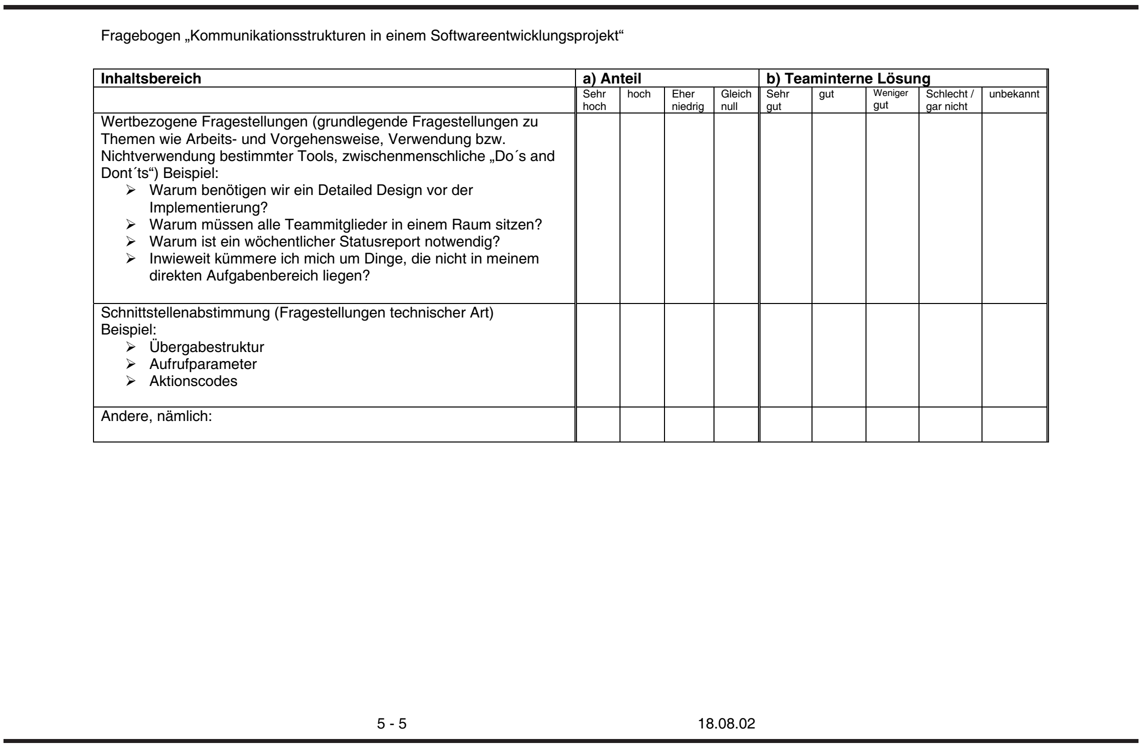
3. Fragestellungen / Problembereiche teamintern

a) Bei der teaminternen Kommunikation: Welche Fragestellungen nahmen welchen Anteil ein ?
b) Wie gut konnte der genannte Problembereich rein teamintern (ohne Absprachen mit Mitarbeitern außerhalb des eigenen Teams) gelöst werden ?
(„sehr gut“ in der ersten Zeile bedeutet bspw., daß z.B. technische Fragestellungen so gut wie immer innerhalb des Teams, also ohne Hinzuziehung des Architekturteams, gelöst werden konnten.)

Inhaltsbereich	**a) Anteil**				**b) Teaminterne Lösung**				
	Sehr hoch	hoch	Eher niedrig	Gleich null	Sehr gut	gut	Weniger gut	Schlecht / gar nicht	unbekannt
Technische Fragestellungen Beispiel: „wie funktioniert der Codesloader?“									
Funktionale oder fachliche Fragestellungen Beispiel: „sollen / müssen die Daten bei einer bestimmten Aktion gespeichert werden oder nicht?“									
Aufgabenverteilung / Organisation Beispiel: ➢ Wer macht wann was? ➢ Wer macht die Abstimmung mit dem Fachbereich? ➢ Wer stellt die Zeitplanung auf?									

4 - 4 18.08.02

Figure A.4: Interview guide page 4

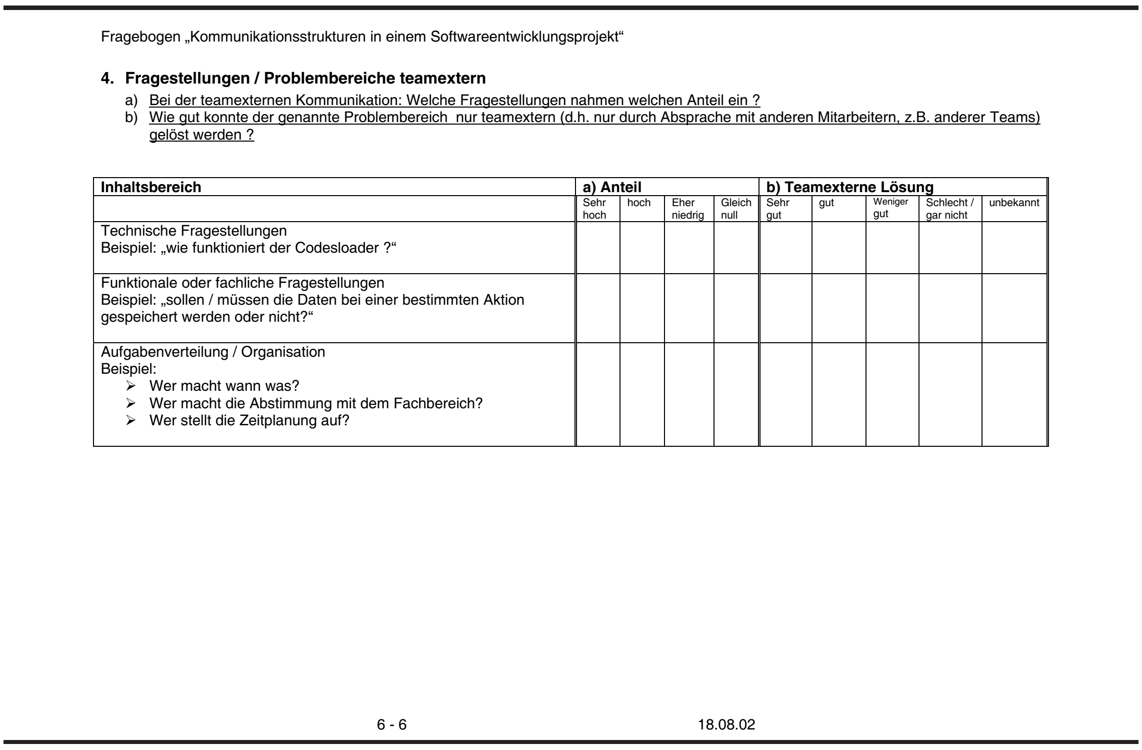


Fragebogen „Kommunikationsstrukturen in einem Softwareentwicklungsprojekt“

Inhaltsbereich	**a) Anteil**				**b) Teaminterne Lösung**				
	Sehr hoch	hoch	Eher niedrig	Gleich null	Sehr gut	gut	Weniger gut	Schlecht / gar nicht	unbekannt
Wertbezogene Fragestellungen (grundlegende Fragestellungen zu Themen wie Arbeits- und Vorgehensweise, Verwendung bzw. Nichtverwendung bestimmter Tools, zwischenmenschliche „Do´s and Dont´ts“) Beispiel: ➢ Warum benötigen wir ein Detailed Design vor der Implementierung? ➢ Warum müssen alle Teammitglieder in einem Raum sitzen? ➢ Warum ist ein wöchentlicher Statusreport notwendig? ➢ Inwieweit kümmere ich mich um Dinge, die nicht in meinem direkten Aufgabenbereich liegen?									
Schnittstellenabstimmung (Fragestellungen technischer Art) Beispiel: ➢ Übergabestruktur ➢ Aufrufparameter ➢ Aktionscodes									
Andere, nämlich:									

5 - 5 18.08.02

Figure A.5: Interview guide page 5



Fragebogen „Kommunikationsstrukturen in einem Softwareentwicklungsprojekt“

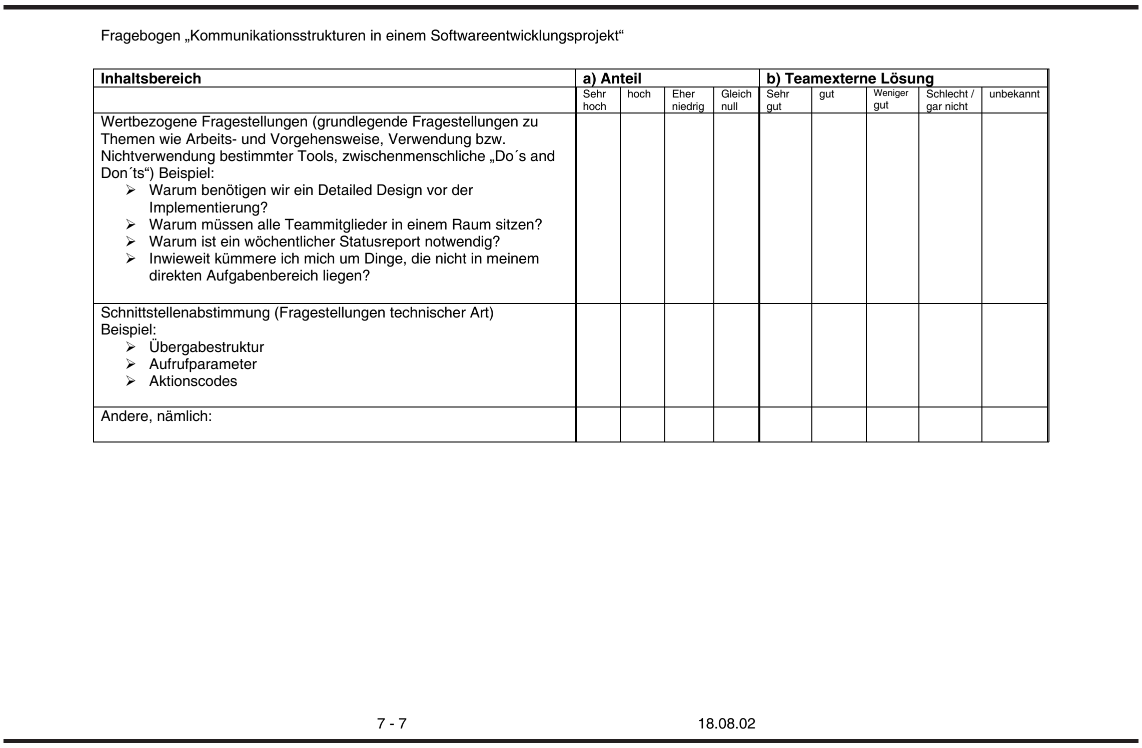
4. Fragestellungen / Problembereiche teamextern

a) Bei der teamexternen Kommunikation: Welche Fragestellungen nahmen welchen Anteil ein ?
b) Wie gut konnte der genannte Problembereich nur teamextern (d.h. nur durch Absprache mit anderen Mitarbeitern, z.B. anderer Teams) gelöst werden ?

Inhaltsbereich	**a) Anteil**				**b) Teamexterne Lösung**				
	Sehr hoch	hoch	Eher niedrig	Gleich null	Sehr gut	gut	Weniger gut	Schlecht / gar nicht	unbekannt
Technische Fragestellungen Beispiel: „wie funktioniert der Codesloader ?“									
Funktionale oder fachliche Fragestellungen Beispiel: „sollen / müssen die Daten bei einer bestimmten Aktion gespeichert werden oder nicht?“									
Aufgabenverteilung / Organisation Beispiel: ➢ Wer macht wann was? ➢ Wer macht die Abstimmung mit dem Fachbereich? ➢ Wer stellt die Zeitplanung auf?									

6 - 6 18.08.02

Figure A.6: Interview guide page 6

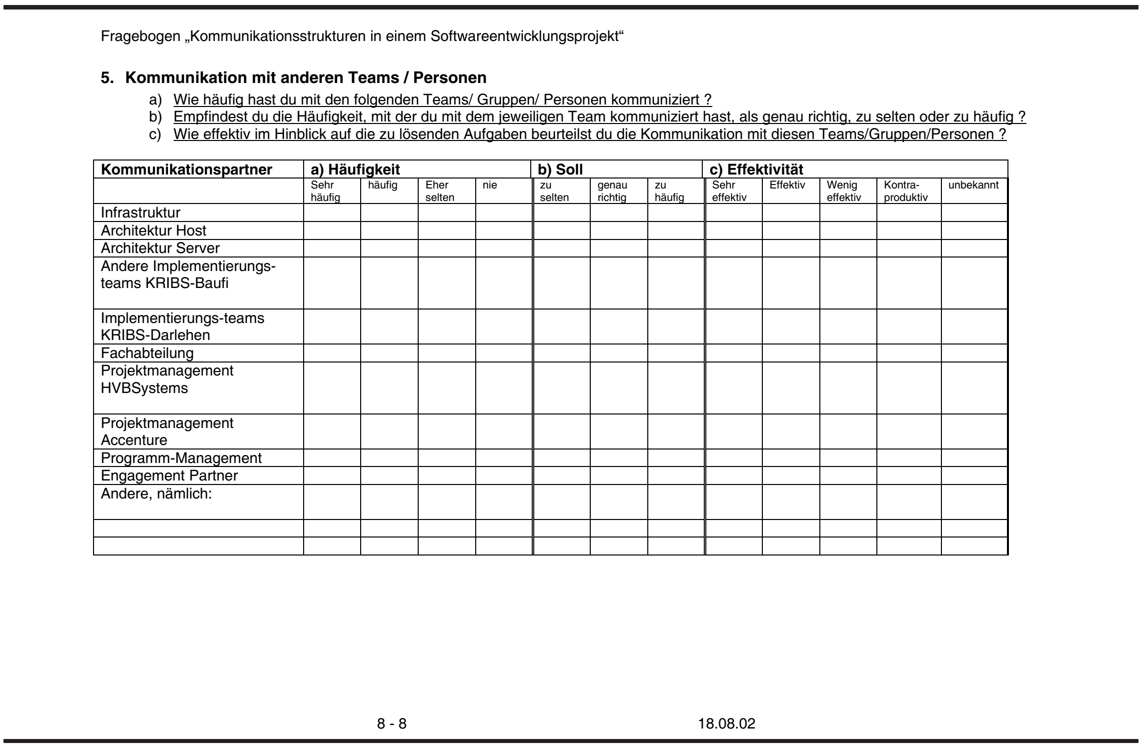


Fragebogen „Kommunikationsstrukturen in einem Softwareentwicklungsprojekt“

Inhaltsbereich	**a) Anteil**				**b) Teamexterne Lösung**				
	Sehr hoch	hoch	Eher niedrig	Gleich null	Sehr gut	gut	Weniger gut	Schlecht / gar nicht	unbekannt
Wertbezogene Fragestellungen (grundlegende Fragestellungen zu Themen wie Arbeits- und Vorgehensweise, Verwendung bzw. Nichtverwendung bestimmter Tools, zwischenmenschliche „Do´s and Don´ts“) Beispiel: ➢ Warum benötigen wir ein Detailed Design vor der Implementierung? ➢ Warum müssen alle Teammitglieder in einem Raum sitzen? ➢ Warum ist ein wöchentlicher Statusreport notwendig? ➢ Inwieweit kümmere ich mich um Dinge, die nicht in meinem direkten Aufgabenbereich liegen?									
Schnittstellenabstimmung (Fragestellungen technischer Art) Beispiel: ➢ Übergabestruktur ➢ Aufrufparameter ➢ Aktionscodes									
Andere, nämlich:									

7 - 7 18.08.02

Figure A.7: Interview guide page 7



Fragebogen „Kommunikationsstrukturen in einem Softwareentwicklungsprojekt“

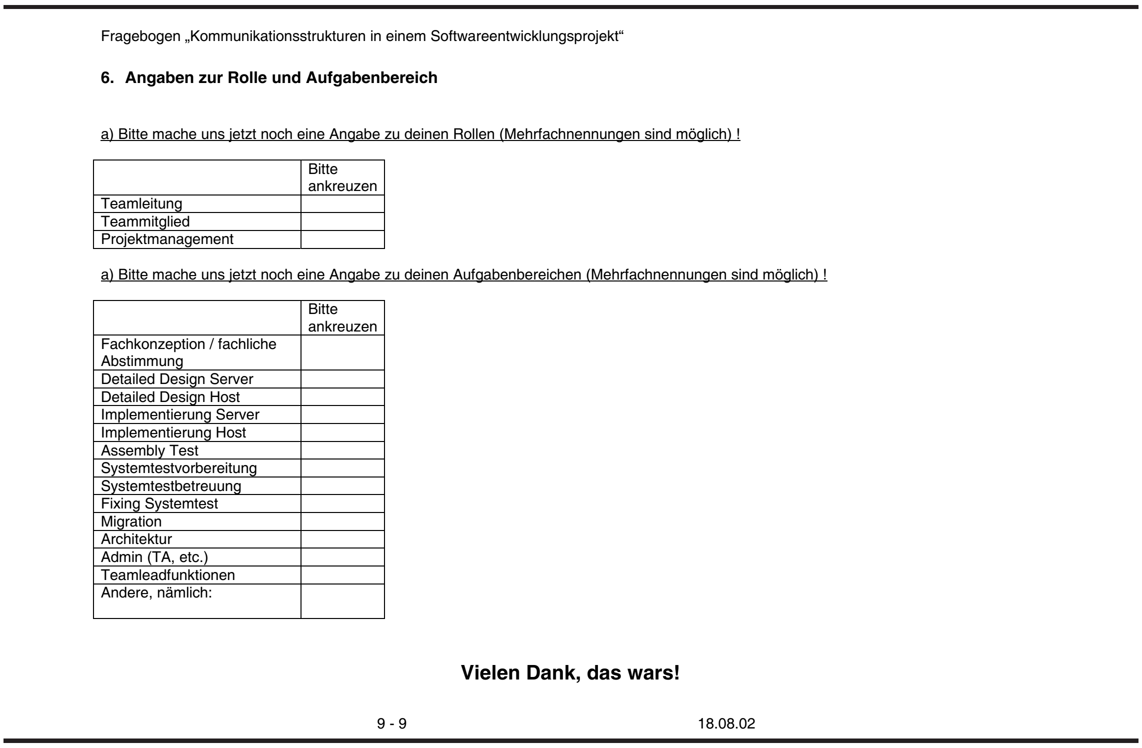
5. Kommunikation mit anderen Teams / Personen

a) <u>Wie häufig hast du mit den folgenden Teams/ Gruppen/ Personen kommuniziert ?</u>
b) <u>Empfindest du die Häufigkeit, mit der du mit dem jeweiligen Team kommuniziert hast, als genau richtig, zu selten oder zu häufig ?</u>
c) <u>Wie effektiv im Hinblick auf die zu lösenden Aufgaben beurteilst du die Kommunikation mit diesen Teams/Gruppen/Personen ?</u>

Kommunikationspartner	**a) Häufigkeit**				**b) Soll**			**c) Effektivität**				
	Sehr häufig	häufig	Eher selten	nie	zu selten	genau richtig	zu häufig	Sehr effektiv	Effektiv	Wenig effektiv	Kontra-produktiv	unbekannt
Infrastruktur												
Architektur Host												
Architektur Server												
Andere Implementierungs-teams KRIBS-Baufi												
Implementierungs-teams KRIBS-Darlehen												
Fachabteilung												
Projektmanagement HVBSystems												
Projektmanagement Accenture												
Programm-Management												
Engagement Partner												
Andere, nämlich:												

8 - 8 18.08.02

Figure A.8: Interview guide page 8



Fragebogen „Kommunikationsstrukturen in einem Softwareentwicklungsprojekt"

6. Angaben zur Rolle und Aufgabenbereich

a) Bitte mache uns jetzt noch eine Angabe zu deinen Rollen (Mehrfachnennungen sind möglich) !

	Bitte ankreuzen
Teamleitung	
Teammitglied	
Projektmanagement	

a) Bitte mache uns jetzt noch eine Angabe zu deinen Aufgabenbereichen (Mehrfachnennungen sind möglich) !

	Bitte ankreuzen
Fachkonzeption / fachliche Abstimmung	
Detailed Design Server	
Detailed Design Host	
Implementierung Server	
Implementierung Host	
Assembly Test	
Systemtestvorbereitung	
Systemtestbetreuung	
Fixing Systemtest	
Migration	
Architektur	
Admin (TA, etc.)	
Teamleadfunktionen	
Andere, nämlich:	

Vielen Dank, das wars!

9 - 9 18.08.02

Figure A.9: Interview guide page 9

Appendix B

ABBREVIATIONS AND GLOSSARY

B.1 ABBREVIATIONS

ATD	Actuals To Date
API	Application Programming Interface
BB	Blackboard
BBS	Bulletin Board Services
CDS	Collaborative Development System
CMM	Capability Maturity Model
CR	Change Request
DCBA	Distributed Concurrent Blackboard Architecture
DOM	Document Object Model
DTD	Data Type Definition
ETC	Estimated To Complete
CSCL	Computer-Supported Collaborative Learning
CSCW	Computer-Supported Cooperative Work
GDT	Geographically Dispersed Team
GSD	Global Software Development
GSE	Global Software Engineering
GUI	Graphical User Interface
HCI	Human-Computer Interface
IDE	Integrated Development Environments
IRC	Internet Relay Chat
ISDN	Integrated Services Digital Network
J2EE	Java2 Enterprise Edition
J2SE	Java2 Standard Edition
JDK	Java Development Kit
JSDK	Java Server Development Kit
JSP	JavaServer Pages
JRE	Java Runtime Evironment
JFC	Java Foundation Classes
JVM	Java Virtual Machine
JWS	Java Web Server
KS	Knowledge Source
KX	Knowledge eXchange
LAN	Local Area Network
MVC	Model-View-Controller
NIS	Network Information Service
NUS	National University of Singapore
PMW	Project Management Workbenches
QOC	Question, Option, Criteria
QoS	Quality of Service
RMI	Remote Method Invocation
ROI	Return on Investment
RPC	Remote Procedure Call
SAX	Simple API for XML
SBBA	Single BlackBoard Architecture
SDK	Servlet Development Kit
SIR	System Investigation Request
SMIL	Synchronized Multimedia Integration Language
SQL	Structured Query Language
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TUM	Technische Universität München
UML	Unified Modelling Language
URI	Uniform Remote Identifier
URL	Uniform Remote Locator
XML	Extensible Markup Language
XP	eXtreme Programming
XSL	Extensible Stylesheet Language
WAN	Wide-Area Network
WWW	World Wide Web

B.2 GLOSSARY

Computer-Supported Cooperative Work.

The computer-supported cooperative work (CSCW) community has been addressing issues of shared development of knowledge artifacts for many years. The community has at least one annual conference in the United States. Groupware is sometimes used as a synonym for CSCW, and Lotus Notes often appears to be the defining CSCW application – even though there are other groupware products.

The term *artifact-based collaboration* is often used in association with CSCW technology because the result of the activity is an artifact – for example, a document authored by many people.

Recent developments in corporate intranets are likely to dramatically increasing the level of interest in CSCW, as IP-based technologies replace or complement proprietary products like Notes.

→ See also shared [work]spaces.

Data Mining.

The process of discovering new valuable coherences, patterns, and trends through the analysis of large amounts of information using pattern-recognition as well as statistical and mathematical methods (Erick Brethenoux, Gartner Group).

Information & Knowledge Mining.

Intellectual assets – especially in computer-supported resources – can be tapped through information mining (or knowledge mining), which typically involves using technology to extract (additional) value from those assets. Knowledge and information mining, however, is not just a matter of technology. It often takes a combination of technology and marketing talents to extract value from the intellectual assets of the company.

The process of information mining begins with finding and managing the right data sources. The systems and marketing research department presides over an impressive array of information feeds and repositories, from vast databases of trials of information (used to position the company's products) to data purchased from information vendors that cater to the pharmaceuticals industry. Sources of competitive intelligence and general information about the healthcare industry include an organization's own sales force, news feeds, online search services and traditional library resources as well as Internet discussion groups and World Wide Web sites devoted to specific diseases.

Intellectual Assets, Capital.

Intellectual assets and intellectual capital (also *intangible assets* and *invisible assets*) generally refer to an organization's recorded information (and, increasingly, human talent itself), and often in particular to patents. The terms reflect the understanding that information is a growing part of every company's assets, and that such information is typically either inefficiently warehoused or simply lost, especially in large, physically dispersed organizations.

Intellectual capital is becoming the corporate country's most valuable asset and can be its sharpest competitive weapon. The challenge is to find what you have and use it.

→ See also corporate repositories, information mining, knowledge mapping.

Knowledge Acquisition.

Knowledge acquisition is the primary job function of a knowledge engineer, for instance a *meeting champion* in the iBistro context. Knowledge acquisition traditionally consists of "the reduction of a large body of knowledge to a precise set of facts and rules" and is associated with expert systems technology (see Brian R. Gaines and Mildred L. G. Shaw, "Eliciting Knowledge and Transferring It Effectively to a Knowledge-Based System"). Recently, these functions (and related job description) seem to be making a transition from addressing specific project requirements to meeting broad organizational objectives.

→ See also *knowledge engineers, knowledge base,* and *meeting champions*.

Knowledge Base.

Knowledge base has traditionally referred to the data and "rules of thumb" produced by the knowledge-acquisition and compilation phases of creating an expert system application. This definition is now often broadened to include every imaginable corporate intellectual asset. "The knowledge base is the absolute collection of all expertise, experience and knowledge of those within any organization."

Design Rationale.

Design rationale methods focus on modeling the justification and argumentation behind decisions. The hypothesis is that rationale information is critical when changing a system, as many defects are introduced due to lack of knowledge of the original rationale. Also, it has been found that rational methods facilitate negotiation among different stakeholders by making underlying arguments explicit.

Software Engineering.

Software Engineering is the process of developing quality software; it's methods focus on modeling the system under construction at different levels of abstraction. (See [Bruegge and Dutoit, 1999; Bruegge and Dutoit, 2003].)

Structured Analysis.

Design is the simple and recursive process by which the designer refines a functional description of the problem into smaller subproblems. [Potts, 1996]

Knowledge.

Our perspective is that knowledge is information transformed into capabilities for effective action. In effect, knowledge is action. In spite of the many different definitions of knowledge, the common ground in almost every discussion of knowledge is that raw information in large quantities does not by itself solve business problems, produce value, or enhance competitiveness.

Knowledge Management.

From the economic point of view, knowledge management is a strategy that turns an organization's →intellectual assets – both recorded information and the talents of its members – into greater productivity, new value, and increased competitiveness. It teaches corporations, from managers to employees, how to produce and optimize skills as a collective entity.

Knowledge transfer.

Knowledge transfer is the tactical dimension of converting knowledge requirements into working solutions.

Tacit vs. Explicit Knowledge.

Michael Polanyi in 1966 defined the distinction between explicit knowledge, which can be articulated in formal language and transmitted among individuals, and tacit knowledge, personal knowledge embedded in individual experience and involving such intangible factors as personal belief, perspective, and values [Polanyi, 1966]. Polanyi's distinction is mentioned frequently in discussions of knowledge and knowledge management, and it is one of the key premises underlying assertions about the close connection between knowledge and action.

The basic problem, of course, is that tacit knowledge is rarely recorded and shared in business organizations. . . even though tacit knowledge may be the real key to getting things done.

There are plenty of variations on this terminology, including Jeff Conklin's comparable use of informal knowledge and formal knowledge [Conklin, 1997].

Focal Knowledge.
Corporate Memory, Institutional Memory.

In practice, there may be little or no difference between corporate intellectual assets and corporate memory. Intellectual assets, however, seem to connote existing information in explicit form.

There is an increasing industrial interest in the capitalization of know-how of (geographically) dispersed groups of people in an organization. This know-how may relate to problem solving expertise in functional disciplines (e.g., design, testing, production), experiences of human resources, and project experiences in terms of project management issues (e.g., social and organizational aspects related to the project team), design technical issues (e.g., design rationale, history of solution space explored, concurrent engineering techniques), and lessons learned. The coherent integration of this dispersed know-how in a corporation is called *corporate memory*.

The use of a corporate memory is to enable concurrent engineering, to benefit from previous experiences to improve quality and efficiency of development/service efforts and to support dynamic, opportunistic project management and coordination. For example, a corporate memory may serve (1) to enable and support a gradual change of the way people in an enterprise work from a functional oriented discipline to a project-driven, cross-functional work; (2) to enable management of expertise available in an organization and to stimulate learning (through expertise transfer, cross- project fertilizations, dynamic project teams, and distributed control); and (3) to enable and support the formation of virtual corporations (temporal alliances among enterprises based on key competence).

Object-Oriented Design.

Software design is the process by which a system is divided into a number of objects, each of which stands for some concrete thing in the problem domain and with interfaces to other objects [Potts, 1996]. See →*Design*.

Organizational Memory.

See →*Corporate Memory, Institutional Memory*.

Ontology.

An *Ontology* is an explicit notation of a conceptualization. The term is borrowed from philosophy, where an Ontology is a systematic account of Existence. For AI systems, what "exists" is that which can be representated.

Virtual Team.

A Virtual Team — also known as a *Geographically Dispersed Team* (GDT) — is a group of individuals who work across time, space, and organizational boundaries with links strengthened by webs of communication technology. They have complementary skills and are committed to a common purpose, have interdependent performance goals, and share an approach to work for which they hold themselves mutually accountable. Geographically dispersed teams allow organizations to hire and retain the best people regardless of location. A virtual team does not always mean teleworkers. Teleworkers are defined as individuals who work from home. Many virtual teams in today's organizations consist of employees both working at home and small groups in the office but in different geographic locations.

Knowledge Representation.

Knowledge representation – explicit specification of "knowledge objects" and relationships among those objects – takes many forms, with variations in emphasis and major variations in formalisms.

Knowledge representation allows computers to reconfigure and reuse information that they store in ways not narrowly prespecified in advance.

Concept mapping.

Seems to be rooted primarily in educational techniques for improving understanding, retention, and as an aid to writing.

A concept map is a picture of the ideas or topics in the information and the ways these ideas or topics are related to each other. It is a visual summary that shows the structure of the material the writer will describe.

Semantic networks.

Are often closely associated with detailed analysis of texts and networks of ideas. One of the important ways they are distinguished from hypertext systems is their support of semantic typing of links – for example, the relationship between "murder" and "death" might be described as "is a cause of." The inverse relationship might be expressed as "is caused by."

Semantic networks are a technique for representing knowledge. As with other networks, they consist of nodes with links between them. The nodes in a semantic network represent concepts. A concept is an abstract class, or set, whose members are things that are grouped together because they share common features or properties. The "things" are called instances of the concept. For example, Femur is a concept representing the set of all femurs in the world; John Smith's left femur is an instance of the concept Femur.

Links in the network represent relations between concepts. Links are labeled to indicate which relation they represent. Links are paired to represent a relation and its inverse relation. For example, the concept Femur is related to the concept Upper Leg with the relation has-location. The inverse of has-location is the relation location-of, which relates Upper Leg to Femur.

Hypertext.

Known to most people these days by its implementation in the World Wide Web, is sometimes described as a semantic network with [substantial] content at the nodes. But the content itself – the traditional document model – seems to be the driving organizational force, not the network of links. In most hypertext documents, the links are not semantically typed, although they are typed at times according to the medium of the object displayed by traversing the link.

→ See also *Distributed Hypertext Systems*.

Information modeling.

Information modeling nterests itself in precise specification of the meaning in a text, and in making relationships of meaning explicit – often with the objective of rapid and accurate development of new software applications for business requirements.

Some of the essence of information modeling is expressed in the proceedings of a recent workshop on object-oriented systems:

How do we: elicit requirements from domain experts, formulate a complete and precise specification understandable to both domain experts and developers, and refine it using existing (or possible) implementation mechanisms.

Fuzzy terminology results in fuzzy thinking: precise and explicit definitions are essential for understanding and reuse…

Conceptual indexing.

Conceptual (or "back-of-the-book") indexes are rarely discussed in the same breath as hypertext, conceptual maps, and semantic networks – perhaps because indexers themselves sometimes relish the aura of “black art” surrounding indexing – but the connection is fundamental. Conceptual indexes traditionally map key ideas and objects in a single work:

An index is a structured sequence – resulting from a thorough and complete analysis of text – of synthesized access points to all the information contained in the text. The structured arrangement of the index enables users to locate information efficiently. [p. 4]

Metadata.

is simply information added to a document (or a smaller unit of information) that makes it easier to access and re-use that content. It’s also referred to as simply "data about data." You’ll find metadata in many different forms, including key words in a software help system, the document profile information attached to documents in a document management system, and the classification information in a library card catalog.

There are, of course, distinctions in how these various disciplines and technologies implement metadata – in substance as well as in formalisms. But the value of metadata for critical information is widely accepted as a basic element of knowledge management implementations. In fact, the term metadata has become so popular that it recently merited its own IEEE conference.

There is a strong interest in metadata in the *geographic information systems* (GIS) community – the one concerned with maps, not the technology for graphic representation of the location of corporate intellectual assets. Claritech’s Elise Yoder observes that the “motherlist” for current work on Metadata seems to be “Metadata Resources”.

BIBLIOGRAPHY

[Al-Rawas and Easterbrook, 1996] Al-Rawas, A. and Easterbrook, S. (1996). Communication problems in requirements engineering: A field study. In *Proc. First Westminster Conf. Professional Awareness in Software Engineering*, Univ. Westminster, London.

[Andind K. Dey, 2001] Andind K. Dey, Daniel Salber, G. D. A. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16.

[Anthes, 2003] Anthes, G. H. (2003). Smart rooms: It assisted workspaces can boost design productivity. Technical report, Computerworld.

[Arlow et al., 1999] Arlow, J., Emmerich, W., and Quinn, J. (1999). Literate modelling — capturing business knowledge with the uml. In *J. Bezivin and P.-A. Muller (eds) The Unified Modeling Language: «UML '98»: Beyond the Notation*, volume 1618, pages 189–199, Mulhouse, France. Springer Verlag.

[Bass et al., 1998] Bass, L., Clements, P., and Kazman, R., editors (1998). *Software Architecture in Practice*. Addision Wesley Longman, Inc., Upper Saddle River, NJ.

[Beck, 1999] Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Publishing Corp.

[BIS, 2003] BIS (2003). The new basel capital accord. Technical report, Basel Committee on Banking Supervision.

[Blackburn et al., 2000] Blackburn, J., Scudder, G., and Wassenhove, L. N. V. (2000). Concurrent software development. In *Communications of the ACM*, volume 43, pages 200–214. ACM Press.

[Bly et al., 1993] Bly, S., Harrison, S., and Irwin, S. (1993). Media spaces: Bringing people together in a video, audio, and computing environment. In *Communications of the ACM*, volume 36(1), pages 28–47.

[Boehm et al., 1998] Boehm, B., Egyed, A., Kwan, J., Port, D., Shah, A., and Madachy, R. (1998). Using the WinWin spiral model: A case study. In *IEEE Computer*. University of Southern California.

[Booch, 2000] Booch, G. (2000). The software development team. Technical report, Rational Software.

[Borghoff and Schlichter, 2000] Borghoff, U. M. and Schlichter, J. H. (2000). *Computer-Supported Cooperative Work: Introduction to Distributed Applications*. Springer. also available in german.

[Boulila et al., 2003] Boulila, N., Bruegge, B., and Dutoit, A. H. (2003). D-meeting: an object-oriented framework for supporting distributed modeling of software. In *Proceedings of ICSE 2003*, Portland, Oregon.

[Braun et al., 2001a] Braun, A., Bruegge, B., and Dutoit, A. H. (2001a). Supporting informal requirements meetings. In *7th International Workshop on Requirements Engineering: Foundation for Software Quality. (REFSQ'2001)*, volume 7, Interlaken, Switzerland.

[Braun et al., 2003] Braun, A., Bruegge, B., and Dutoit, A. H. (2003). A software architecture for knowledge acquisition and retrieval in global software development. In *Proceedings of ICSE 2003*, Portland, Oregon.

[Braun et al., 2001b] Braun, A., Bruegge, B., Dutoit, A. H., and Harrer, A. G. (2001b). iBistro: A learning environment for knowledge construction in distributed software engineering courses. In *Proceedings of the International Conference on Computers in Education (ICCE'2001)*, Seoul, Korea.

[Braun et al., 2002a] Braun, A., Bruegge, B., Dutoit, A. H., and Harrer, A. G. (2002a). iBistro: A learning environment for knowledge construction in distributed software engineering courses. In *Proceedings of APSEC 2002; Extended and updated version of [Braun et al., 2001b]*, Queensland, Australia.

[Braun and Harrer, 2000] Braun, A. and Harrer, A. G. (2000). A framework for internet-based distributed learning. In *Proceedings of the International Conference on Computers in Education*, volume 6, Taipei, Taiwan.

[Braun et al., 2002b] Braun, A., Mang, F., Behr, W., Bruegge, B., and Dutoit, A. H. (2002b). A communication field study in a large, team-based consulting project. Submitted to the Proceedings of APSEC 2002.

[Brody et al., 1999] Brody, A. B., Dempski, K. L., Kaplan, J. E., Kurth, S. W., Liongosari, E. S., and Swaminathan, K. S. (1999). Integrating disperate knowledge sources. In *Proc. of the Second International Conference on The Practical Application of Knowledge Management (PAKeM '99)*, London. Accenture.

[Broekstra et al., 2002] Broekstra, J., Klein, M., Decker, S., Fensel, D., van Harmelen, F., and Horrocks, I. (2002). Enabling knowledge representation on the web by extending rdf schema. In *Computer Networks*, volume 39(5), pages 609–634. Elsevier Science B.V.

[Brooks, 1986] Brooks, F. P. (1986). No silver bullet. In H.J.Kugler, editor, *Information Processing 1986: Proceedings of the IFIP Tenth World Computing Conference*, pages 1069–1076, Amsterdam, The Netherlands.

[Brooks, 1995] Brooks, F. P. (1995). *The Mythical Man-month*. Addison Wesley, San Diego, 2nd. edition.

[Bruegge and Dutoit, 1999] Bruegge, B. and Dutoit, A. H. (1999). *Object-Oriented Software Engineering: Conquering Complex and Changing Systems*. Prentice Hall, Upper Saddle River, NJ, 1st. edition.

[Bruegge and Dutoit, 2003] Bruegge, B. and Dutoit, A. H. (2003). *Object-Oriented Software Engineering: Conquering Complex and Changing Systems*. Prentice Hall, Upper Saddle River, NJ, 2nd. edition.

[Bruegge et al., 1999] Bruegge, B., Dutoit, A. H., Kobylinski, R., and Teubner, G. (1999). Transatlantic software development: Teaching distributed software engineering. In *IEEE*, Singapore. IEEE.

[Burbeck, 1987] Burbeck, S. (1987). Application programming in Smalltalk-80: How to use Model-View-Controller (MVC).

[Buschmann et al., 1996] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M., editors (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons Ltd., Chichester, West Sussex, England.

[Buxton et al., 1997] Buxton, W., Sellen, A., and Sheasby, M. (1997). *Interfaces for multiparty videoconferences, Video-mediated communication*. Lawrence Erlbaum Associates.

[Carmel, 1999] Carmel, E. (1999). *Global Software Teams: Collaborating Across Borders and Time Zones*. Prentice Hall, Upper Saddle River, NJ.

[Carver and Lesser, 1992] Carver, N. and Lesser, V. (1992). The evolution of blackboard control architectures. Technical report, CMPSCI Technical Report 92-71.

[Conklin, 1997] Conklin, E. J. (1997). Designing organizational memory: Preserving intellectual assets in a knowledge economy. Glebe Creek, MD: CogNexus Institute.

[Conklin and Begeman, 1988] Conklin, J. and Begeman, M. (1988). gIBIS: A hypertext tool for exploratory policy discussion. In *ACM Transactions on Office Information Systems*, volume 6, pages 303–331.

[Cranefield and Purvis, 1999] Cranefield, S. and Purvis, M. (1999). Uml as an ontology modelling language. In *Proceedings of the Workshop on Intelligent Information Integration*, volume 16. International Joint Conference on Artificial Intelligence (IJCAI-99).

[Curtis et al., 1988] Curtis, B., Krasner, H., and Iscoe, N. (1988). A field study of the software design process for large systems. In *Communications of the ACM*, volume 31(11).

[Damian et al., 2000] Damian, D., Eberlein, A., Shaw, M., and Gaines, B. (2000). Using different communication media in requirements negotiation. In *IEEE Software*, pages 28–36.

[DeMarco and Lister, 1979] DeMarco, T. and Lister, T. (1979). *Peopleware: Productive Projects and Teams*. Dorset House, San Diego, 1st. edition.

[Dourish, 1996] Dourish, P. (1996). *Open Implementation and Flexibility in CSCW Toolkits*. PhD thesis, Departement of Computer Science, University College London.

[Drogoul, 2001] Drogoul, A. (2001). Dess ia - intelligence artificielle distribuée. Technical report, Le Laboratoire d'Informatique de Paris 6.

[Dutoit and Bruegge, 1998] Dutoit, A. and Bruegge, B. (1998). Communication metrics for software development. In *IEEE Transactions on Software Engineering*, volume 24(8).

[Dutoit and Paech, 2001a] Dutoit, A. H. and Paech, B. (2001a). Developing guidance and tool support for use case-based specification. In *Proceedings of the 7th International Workshop on Requirements Engineering: Foundation for Software Quality. (REFSQ'2001)*, volume 7, Interlaken, Switzerland.

[Dutoit and Paech, 2001b] Dutoit, A. H. and Paech, B. (2001b). Rationale management in software engineering. In Chang, S., editor, *Handbook on Software Engineering and Knowledge Engineering*, chapter 1.2. World Scientific Publishing, River Edge, NJ, USA.

[Easterbrook et al., 1994] Easterbrook, S., Finkelstein, A., Kramer, J., and Nuseibeh, B. (1994). Co-ordinating distributed viewpoints: the anatomy of a consistency check. Technical report, Department of Computing, Imperial College, London, UK.

[Ellis et al., 1991] Ellis, C. A., Gibbs, S. J., and Rein, G. L. (1991). Groupware – some issues and experiences. In *Communications of the ACM*, volume 34(1), pages 38–58.

[Erman et al., 1980] Erman, L. D., Hayes-Roth, F., Lesser, V. R., and Reddy, D. R. (1980). The hearsay-ii speech-understanding system: Integrating knowledge to resolve uncertainty. In *ACM Computing Surveys*, volume 12 (2), pages 213–253.

[Fischer et al., 2002] Fischer, M., Hanrahan, P., Kunz, J., and Winograd, T. (2002). Cife interactive information workspace and cife iroom to go. Technical report, Stanford University.

[Forgy and McDermott, 1977] Forgy, C. and McDermott, J. P. (1977). Ops, a domain-independent production system language. *International Joint Conference on Artificial Intelligence (IJCAI-77)*, pages 933–939.

[Fox et al., 2000] Fox, A., Johanson, B., Hanrahan, P., and Winograd, T. (2000). Integrating information appliances into an interactive workspace. In *IEEE Information Appliances*.

[Friedrichs, 1984] Friedrichs, J., editor (1984). *Methoden empirischer Sozialforschung*. Westdeutscher Verlag.

[Fuchs et al., 1978] Fuchs, W., Klima, R., Lautmann, R., Rammstedt, O., and Wienold, H., editors (1978). *Lexikon zur Soziologie*. Westdeutscher Verlag, 2nd edition.

[Gordon, 1999] Gordon, J. L. (1999). Creating knowledge maps by exploiting dependent relationships. Technical report, Applied Knowledge Research Institute, Blackburn College, Feilden Street, Blackburn, England.

[Grady and Caswell, 1987] Grady, R. B. and Caswell, D. L., editors (1987). *Software Metrics: Establishing a Company-wide Program*. Prentice Hall.

[Green, 2002] Green, D. (2002). The J2EE^{TM} tutorial.

[Grinter et al., 1999] Grinter, R., Herbsleb, J., and Perry, D. (1999). The geography of coordination: Dealing with distance in r& d work. In *Communications of the ACM*.

[Gruber, 1992] Gruber, T. R. (1992). Technical report ksl 92-71: A translation approach to portable ontology specifications. Technical report, Knowledge Systems Laboratory, Computer Science Department, Stanford University, Stanford, California 94305.

[Gruber, 1993] Gruber, T. R. (1993). A translation approach to portable ontology specifications. In *Knowledge Acquisition*, volume 5(2).

[Grudin, 1994] Grudin, J. (1994). Cscw: History and focus. *IEEE Computer*, 27(5):16–19.

[Guba and Lincoln, 1985] Guba, E. and Lincoln, Y., editors (1985). *Naturalistic Inquiry*. Sage Publications.

[Henderson-Sellers, 1996] Henderson-Sellers, B., editor (1996). *Object-Oriented Software Metrics*. Prentice Hall.

[Hengstenberg, 2001] Hengstenberg, O. (2001). Video-based capture of rationale information in informal meetings. Diploma thesis.

[Hughes, 1999] Hughes, L. (1999). Magic wall.

[Ikeda et al., 1997] Ikeda, M., Shogo, G., and Mizoguchi, R. (1997). Opportunistic group formation. In *Proceedings of AI-ED 1997*, pages 167–174, Kobe, Japan.

[Johnson, 1999] Johnson, L., editor (1999). *Instructional Uses of Synthetic Agents*, LeMans, France.

[Johnson, 1998] Johnson, P. A. (1998). Nt 5.0 is too much of a good thing. Technical report, LanTimes.

[Kaufman et al., 2002] Kaufman, N., Pullara, J., Davenport, M. G., and Thompson, C. (2002). Insights from the events of september 11th: Is your organisation prepared? Technical report, PriceWaterhouseCoopers.

[Koch, 2003] Koch, M. (2003). Communixx.de: Community. Technical report, TU-München.

[Kotlarsky, 2001] Kotlarsky, J. (2001). Developing internet-based integrated architecture for managing globally distributed software development projects. Technical report, Department of Decision & Information Sciences, Erasmus University, Rotterdam, The Netherlands.

[Kotonya and Sommerville, 1996] Kotonya, G. and Sommerville, I. (1996). Requirements engineering with viewpoints. In *BCS/IEE Software Engineering Journal*, volume 11(1), pages 5–18.

[Kraut and Streeter, 1995] Kraut, R. and Streeter, L. (1995). Coordination in software development. In *Communications of the ACM*, volume 38(3).

[Kruchten, 2000] Kruchten, P. (2000). *The Rational Unified Process: An Introduction*. Addison-Wesley Publishing Corp.

[Lindvall et al., 2002] Lindvall, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Shull, F., Tesoriero, R., Williams, L., and Zelkowitz, M. (2002). Empirical findings in agile methods. *Proceedings Extreme Programming and Agile Methods - XP/Agile Universe 2002: Second XP Universe and First Agile Universe Conference*, 2418:197–207.

[Lyons, 1991] Lyons, T. (1991). *Network Computing System Tutorial.* Prentice Hall, Englewood Cliffs, NJ.

[MacLean et al., 1996] MacLean, A., Young, R. M., Bellotti, V. M., and Moran, T. P. (1996). *Questions, Options, and Criteria: Elements of Design Space Analysis*, chapter 3, pages 53–106. Design Rationale: Concepts, Techniques, and Use. Lawrence Erlbaum Associates, Hillsdale, NJ, first edition.

[McManus, 1992] McManus, J. W. (1992). *Design and Analysis Techniques for Concurrent Blackboard Systems.* PhD thesis, The College of William and Mary in Virginia.

[Melnik, 2000] Melnik, S. (2000). Uml in rdf. Technical report, Stanford University.

[Moran et al., 1999] Moran, T., van Melle, B., and Saund, E. (1999). *Walls at Work – Physical and Electronic Walls in the Workplace*, pages 191–208. Deutsche Verlags Anstalt, Stuttgart.

[Moran and Dourish, 2001] Moran, T. P. and Dourish, P., editors (2001). *Context-Aware Computing*, volume 16.

[MySQL, 2003] MySQL (2003). The mysql open source database website. Technical report, MySQL Organization.

[Nakanishi et al., 1999] Nakanishi, H., Yoshida, C., Nishimura, T., and Ishida, T. (1999). Freewalk: A 3d virtual space for casual meetings. *IEEE Micro*.

[Nii, 1986a] Nii, H. P. (1986a). Part 1– blackboard systems: The blackboard model of problem solving and the evolution of blackboard architectures. In *AI Magazine*, volume 7(2), pages 38–53.

[Nii, 1986b] Nii, H. P. (1986b). Part 2– blackboard application systems and a knowledge engineering perspective. In *AI Magazine*, volume 7(3), pages 82–107.

[Nii et al., 1982] Nii, H. P., Feigenbaum, E., Anton, J., and Rockmore, A. J. (1982). Signal-to-symbol transformation: Hasp/ slap case study. In *AI Magazine*, volume 3(1), pages 23–35.

[Nikitsch, 2003] Nikitsch, F. (2003). Design and implementation of a shared knowledge repository for ibistro. Diploma thesis.

[OMG and Meta Group, 2000] OMG and Meta Group (2000). Meta data coalition home page. Technical report, Object Management Group (OMG).

[Perlis et al., 1981] Perlis, A. J., Sayward, F., and Shaw, M., editors (1981). *Software Metrics: A Rigorous and Practical Approach*. MIT Press.

[Picot et al., 1996] Picot, A., Reichwald, R., and Wigand, R. T. (1996). *Grenzenlose Unternehmung*. Gabler.

[PictureTel Corporation, 2001] PictureTel Corporation (2001). PictureTel videoconferencing.

[Polanyi, 1966] Polanyi, M. (1966). *The Tacit Dimension*. Routledge & Kegan Paul, London.

[Potts, 1996] Potts, C. (1996). *Supporting Software Design: Integrating Design Methods and Design Rationale*, chapter 10, pages 295–321. Design Rationale: Concepts, Techniques, and Use. Lawrence Erlbaum Associates, Hillsdale, NJ, first edition.

[Pribilla et al., 1996] Pribilla, P., Reichwald, R., and Goecke, R. (1996). *Telekommunikation im Management oder Startegien für den Globalen Wettbewerb*. Schäffer-Poeschel Verlag Stuttgart.

[Rheingold, 2000] Rheingold, H. (2000). *The Virtual Community: Homesteading on the Electronic Frontier*. MIT Press.

[Rittel and Weber, 1973] Rittel, H. and Weber, M. (1973). Dilemmas in a general theory of planning. In *Policy Science*, volume 4, pages 155–169.

[Rumbaugh et al., 1998] Rumbaugh, J., Jacobson, I., and Booch, G. (1998). *The Unified Modeling Language Reference Manual*. Addision Wesley Longman, Inc., Reading Mass.

[Rumbaugh et al., 1999] Rumbaugh, J., Jacobson, I., and Booch, G. (1999). *The Unified Modeling Language Reference Manual*. Addison Wesley Longman, San Diego.

[Saund, 1998] Saund, E. (1998). Image mosaicing and a diagrammatic user interface for an office whiteboard scanner. Technical report, Xerox PARC.

[Seaman and Basili, 1998] Seaman, C. and Basili, V. (1998). Communication and organization: An empirical study of discussion in inspection meetings. In *IEEE Transactions on Software Engineering*, volume 24(6).

[Seaman, 1999] Seaman, C. B. (1999). Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, 25(4):557–572.

[Sellen, 1995] Sellen, A. (1995). Remote conversations: the effects of mediating talk with technology. *Human-Computer Interaction*, 10(4):401–444.

[SkillSoft, 2002] SkillSoft (2002). Project management fundamentals. Technical report, SkillSoft – The eLearning Solutions Company.

[SMART, 2000a] SMART (2000a). Hawkeye.

[SMART, 2000b] SMART (2000b). SMART Board.

[Smith et al., 1990] Smith, S. F., Ow, P. S., Muscettola, N., Potvin, J.-Y., and Matthys, D. C. (1990). An integrated framework for generating and revising factory schedules. In *Journal of the Operational Research Society*, volume 41(6), page 539–552.

[SourceForge, 2001] SourceForge (2001). SourceForge: SourceForge services. Technical report, SourceForge.

[Sun Microsystems, 1998] Sun Microsystems (1998). Java remote method invocation specification, revision 1.50, jdk 1.2.

[Sun Microsystems, 2003] Sun Microsystems (2003). The javaspaces technology.

[Sutcliffe and Ryan, 1998] Sutcliffe, A. and Ryan, M. (1998). Experience with scram, a scenario requirements analysis method. In *Proc. of the 3rd International Conference on Requirements Engineering*, pages 164–171.

[Tanenbaum, 2001] Tanenbaum, A. S. (2001). *Modern Operating Systems*. Prentice Hall, 2nd. edition.

[Taylor and Bogdan, 1984] Taylor, S. and Bogdan, R. (1984). *Introduction to Qualitative Research Methods*. John Wiley and Sons, New York.

[Teufel et al., 1995] Teufel, S., Sauter, C., and Mühlherr, T. (1995). *Computerunterstützung für die Gruppenarbeit*. Addison-Wesley Publishing Company, Bonn/ Paris.

[W3C, 1998a] W3C (1998a). SMIL. Technical report, World Wide Web Consortium.

[W3C, 1998b] W3C (1998b). XML. Technical report, World Wide Web Consortium.

[W3C, 1998c] W3C (1998c). XSL and XSLT. Technical report, World Wide Web Consortium.

[Want et al., 1992] Want, R., Hopper, A., Falcao, V., and Gibbons, J. (1992). The active badge location system. In *ACM Transactions on Information Systems*, volume 10(1), pages 91–102.

[Weinberg, 2001] Weinberg, G. (2001). *The Secrets of Consulting: A Guide to Giving and Getting Advice Successfully*. Dorset House Publishing Company, Incorporated.

[Weiser, 1991] Weiser, M. (1991). The computer for the 21st century. In *Scientific American*, pages 66–75.

[Winograd, 2001] Winograd, T. (2001). Architectures for context. *HCI Journal Special Issue on Context-aware Computing*, 16.