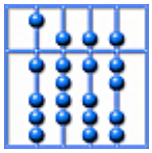
Technische Universität München
Institut für Informatik



A Framework for Distributed Collaborative Software Design Meetings

Dissertation

Naoufel ben Ahmed Boulila

Institut für Informatik
Der Technischen Universität München
Lehrstuhl für Informatik I

A Framework for Distributed Collaborative Software Design Meeting

Naoufel ben Ahmed Boulila

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. Hans-Joachim Bungartz
Prüfer der Dissertation:	
	1. Univ. -Prof Bernd Brügge. Ph.D.
	2. Univ. -Prof. Dr. Johann Schlichter

Die Dissertation wurde am 11.07.2005 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 10.10.2005 angenommen.

Abstract

Over the last decade, software development in large enterprise organizations has witnessed an increasing tendency towards globalization and global outsourcing, seeking lower costs and access to skilled resources.

Globalization has made distributed software development collaborating at the same time over multiple geographical sites a major challenge because of several issues, including language and cultural differences, communication across temporal and spatial distances, trust factors, and the lack of shared contextual awareness. These issues have been identified through several studies conducted on globally distributed software development. However, none of these studies have considered global and distributed real-time group software design.

The distribution of software design introduces new aspects of cooperative work in which a greater emphasis is placed upon effective methods and the technological support of the software development process. Most software development activities and in particular software brainstorming and design, are carried out in a group context, such as collaborative meetings, rather then individually. Globalization has further accentuated the necessity of group collaboration.

This research focuses on the specific problem of distributed collaborative brainstorming and design for object-oriented software analysis and design activities. In particular, the thesis explores the requirements for synchronous collaborative UML modeling and the accumulation of associated design knowledge through iterative distributed team meetings.

The main hypothesis of this thesis is that distributed software brainstorming and design meetings are feasible in distributed sites collaborating at the same time.

The thesis hypothesis is evaluated in a case study developing a software architecture called SCOOP (Synchronous Collaborative Object-Oriented Process) to support distributed collaborative conceptual software modeling meetings. The architecture uses techniques of object-oriented components and a contract-based style. Applications such as distributed mind mapping or distributed collaboration over concept maps can reuse the SCOOP framework by instantiating and extending its components.

A formative approach was adapted and used to conduct an experimental evaluation of the case studies to validate the research hypothesis. The use of the formative approach was facilitated with a bootstrapping concept that led the development of a new agile-based design style method called *BID* (Bootstrapping Incremental Design), which was elaborated and used in developing the SCOOP framework.

A reference implementation of SCOOP has been developed based on the BID approach and tested in many distributed software design meetings.

Kurzfassung

Im Laufe des letzten Jahrzehnts hat der Bereich der Softwareentwicklung in Großunternehmen auf der Suche nach Kostenreduzierung und Zugang zu qualifiziertem Personal eine zunehmende Tendenz zur Globalisierung und weltweiten Auslagerung erfahren.
Die Globalisierung hat die gemeinsame verteilte Softwareentwicklung an mehreren Standorten, die auf der ganzen Welt verteilt sind, vor verschiedene Herausforderungen gestellt. Dazu gehören unter anderem sprachliche und kulturelle Unterschiede, die Kommunikation über Zeitzonen und große Entfernungen hinweg, Vertrauensfaktoren und das Fehlen von gemeinsamem Kontextbewusstsein.
Diese Faktoren wurden durch mehrere Studien festgestellt, die global verteilte Softwareentwicklung untersucht haben, allerdings hat keine dieser Studien global verteilten Softwareentwurf in Echtzeit berücksichtigt.
Die Verteilung des Softwareentwurfs führt neue Kooperationsaspekte ein, bei denen effektive Methoden und die technische Unterstützung des Softwareentwicklungsprozesses an Bedeutung gewinnen.

Diese Arbeit konzentriert sich auf das spezielle Problem der verteilten Ideensammlungen und Entwurfsbesprechungen bei Analyse und Entwurf objektorientierter Software.
Insbesondere untersucht die Dissertation die Anforderungen an synchrone gemeinsame UML-Modellierung und die Sammlung des dabei entstehenden Entwurfswissens durch iterative verteilte Teambesprechungen.
Die zentrale Hypothese der Dissertation ist die Annahme, dass verteilte, gleichzeitige Ideensammlungen und Entwurfsbesprechungen über verschiedene Standorte hinweg machbar sind.
Die Hypothese wird in einer Fallstudie überprüft, in der eine Softwarearchitektur namens SCOOP (Synchronous Collaborative Object-Oriented Process) entwickelt wird, die verteilte konzeptionelle Entwurfsbesprechungen unterstützt. Die Architektur setzt objektorientierte Komponenten und einen vertragsbasierten Stil ein. Anwendungen wie verteilte Mind-Maps oder verteilte Zusammenarbeit über Concept-Maps können das SCOOP-Framework wiederverwenden, indem sie dessen Komponenten instanziieren und erweitern.
Mit Hilfe eines angepassten formativen Ansatzes wurde die Fallstudie selbst experimentell evaluiert, um die Hypothese weiter zu überprüfen.
Der formative Ansatz wurde durch ein Bootstrap-Konzept unterstützt, das die Entwicklung einer neuen agilen Entwurfsmethode BID (Bootstrapping Incremental Design) einleitete, die bei der Entwicklung von SCOOP eingesetzt wurde. Eine Referenzimplementierung von SCOOP wurde auf der Basis des BID-Ansatzes entwickelt und in einigen verteilten Softwareentwurfs-Besprechungen getestet.

Acknowledgment

This is the only place in the dissertation where one does not tell about the dissertation but about the people who made it possible to shape the dissertation.

I thank God for everything, including the direction to arrive to this task, the strength to finish it, for all the good ideas and insights, and for giving me the chance to know the following great people:

Prof. Bernd Brügge, PhD. who believed in me and my ability to do scientific work from the very first meeting. I deeply thank him not only for helping me overcoming many bureaucratic issues but also for the continual support and inestimable guidance throughout the years i spent in his chair. I am forever thankful to prof. Brügge for the countless hours of virtual and real "brainstorming meetings" that took place even during the holidays, the sundays, over the phone, in restaurants and everywhere we could have a meeting.

I would like to thank my second reviewer, Prof. Dr. Johann Schlichter, for his support, the careful and precise comments, and the valuable review on the final draft.

I thank Fr. Dr. Reiser Angelika who was my first contact with TUM and gave me the right address to start with and showed me the right persons to contact.

I thank Prof. Dr. Arndt Bode for his support during the recognition process of my diploma.

I thank Prof. Rudolf Bayer PhD. for hosting my 6 months stay in his chair for further development of a FORWISS project.

I thank Prof. Dr. Christoph Zenger for letting me using his silicon graphics machines for 3 months, and for accepting to be my examiner for the theoretische informatik pillar.

I would like to acknowledge the assistance and guidance of Allen Dutoit, Ph.D., for being always available to all of us, for the long calls and discussions over the phone even in the weekend, for the moral support, and for being the co-author of most of the publications.

I also thank the following great people:
Tom Ellman, PhD (Vassar College NY) and Andrea Zisman PhD (city university london), and Miguel Baptista Nunes, PhD (the university of Sheffield) for their valuable feedback and support during the two doctoral symposiums i attended.

I thank Diane H.Sonnenwald, PhD (Göteburg University) for her support and for lending me her printed dissertation.
I thank Bonnie John, PhD. (CMU) for her support and for confirming that there was not any formal description of the formative approach yet:)

I thank Mark Roseman for being available to support me with feedback.

I also want to thank my fellow colleagues in the chair of Applied Software Engineering and especially the members of the Global Software Engineering (GSE) research group, and all the staff, Dr. Christian Herzog, Fr. Monika Markl, Fr. Helma Schneider, and Fr.Uta Weber for their valuable support.

A very special thank to Frau Barbara Kalter, for her support in solving the many bureaucratic issues in many ways.

I thank Donald Arthur for the proof reading and the many suggestions for improving the writing in english.

I thank Sghaier Guizani (Universite de Quebec) for the feedback on earlier draft of the thesis.

I deeply thank Christian Quinto and his family for the continual support since my early coming to Munich. I owe them much.
I also thank Bernhard Schwab (Fujitsu-Siemens) for the his valuable support during many years.
A special thank to my colleague Michael Nagel for the SAP-days in Wacker chemie and for his support in many situations.

Finally i deeply thank my wife and family for their continual support, and i apologize for not having enough time for them.

Table Of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Over the last decade software development in large organizations has witnessed an increasing tendency towards globalization and outsourcing, aiming at decreasing costs and facilitating access to skilled resources.

Several studies have been conducted on globally distributed software development [92, 91, 94, 67, 68, 118, 55, 56] and have identified several issues, which are widespread across projects, including language and cultural differences, communication across temporal and spatial distances, trust factors, and the lack of shared contextual awareness. This way, globalization has made software development distributed over multiple geographical sites difficult and challenging.

Distributed software development introduces new aspects of cooperative work in which a greater emphasis is placed upon effective methods and the technological support of the software development process. Most software development activities are carried out in a group context, such as collaborative meetings, rather then individually. Even coding can be performed by a group of two developers in distributed settings as well, such as in a distributed version of XP [2].

Globalization has further accentuated the necessity for group collaboration.

While several researchers have been able to conduct distributed formal reviews and meetings in several distributed projects [6], distributed synchronous problem solving, such as brainstorming, the development of an architecture or specifying requirements, has been much more limited.

Model-based software development is complex; Software development activities deal with the complexity by constructing and validating models of the application domain. Models are important artifacts used for communication within the organizations, and among developers and stakeholders as well. Constructing correct, complete, consistent, and unambiguous models and artifacts calls for the involvement of multiple stakeholders, customers, developers and even organizations.

The only modes of collaboration that have worked so far have been the asynchronous interaction among a small group of people who knew each other beforehand [120]. Current research on synchronous software development across multiple sites is limited to few experiences, e.g. Damian [29] investigated support for distributed requirement engineering; L. Brothers has investigated distributed code inspection [18], Gaoyan et al.[46] have investigated distributed code debugging; finally Dewan [114] presented an environment for concurrent software development that could be used in distributed settings.

The possible geographical dispersion of the organizations makes collaborative software development more complex [39]. Little has been done to support distributed object-oriented model-driven development. It is our thesis that efforts in supporting distributed software development meetings improve distributed development. Distributed meetings typically play a critical role in teamwork, during which large amount of implicit knowledge is exchanged through negotiation and conflict resolution.

Designing models through synchronous meetings is a communication-intensive activity, in which much emphasis is placed on brainstorming ideas and intensive discussions of different design alternatives. Moreover, group communication and knowledge produced during distributed meetings deserve to be

considered as important artifacts aligned with the software architecture and the requirements that the group is developing. This can happen many ways, capturing the design rationale, short-term and long-term collaborative support that involves the sharing of ideas, artifacts, knowledge, and group memory in real-time. Therefore, issues like group awareness, floor control, and multi-user interfaces as well as user-system interactions are crucial and have to be considered a solution. These issues lays the ground for the specification and the design of the CSCW framework to support distributed brainstorming and software design.

In [23] Clever states that on the one hand, most of the existing CSCW frameworks are ad hoc developed and activity-centered (e.g. workflow management, coauthoring systems,...). They make little or no use of elaborate components, and have taken pragmatic approaches in dealing with CSCW key aspects like group awareness information, group activity, and floor control policies. They are tightly coupled with the activity undertaken. On the other hand, Clever states that most of the conceptual CSCW frameworks are too general to be of any practical use or too restricted to a particular application. This is mostly caused by the complexity of the application and solution domains. Nierstrasz and Tsichritzis [111] state that object-oriented techniques promise to cope with that complexity.

SCOOP is an object-oriented framework that supports distributed synchronous brainstorming and software design [107]. The development of SCOOP object model served at capturing the concepts present in the application domain under investigation such as group brainstorming, group awareness, floor control, knowledge and rationale management. SCOOP is composed of several sub-object models for the activity, communication, collaboration, and coordination.

Although this is not the main concern of this thesis, we have used the object-oriented techniques for several reasons. Among these, first, in using object-oriented analysis and design techniques we seek to build a reliable design and a sturdy architecture for distributed development. Secondly, we aim to support the developers who would reuse SCOOP framework specifications. This is

why applications derived from SCOOP can benefit from the object-oriented reuse in redefining their own behavior.

The main idea in the design and implementation of the SCOOP framework is to map the identified concepts of the application domain into object components. This is why the abstract models of the activity, the collaboration, the communication, the coordination, and the rationale management are represented by object components. For example, different implementations of awareness information aspects can be used to show different behaviors according to the application needs without affecting the underlined activity of the group. Abstracting a key aspect of the application domain as a set of objects, makes possible the alternative use of different implementations of the key aspect within the framework even at run time. Applications (re)using SCOOP may profit from this possibility to load different implementations of object components representing a key aspect at run-time. A mediator coordinates the inter-object components interaction and communication. Similar approaches were used in OVAL [89] by Malone and in OOActSM by Teege [139] which is centered around the concept of activity. Both aim at providing an integrated framework for CSCW systems. Although these approaches are already useful, they are too general in use. OOActSM does not make group collaborative aspects like group awareness, floor control or rationale management explicit; these concepts and their logic have to be implemented as activities. OVAL belongs to a category of systems called meta-groupware used to build cooperative systems, which are not necessarily domain-driven systems and have to be refined accordingly[23].

1. Objectives

The objectives of this thesis are manifold. The proposed solutions, the dissertation specific aims are:

- To conduct and evaluate the feasibility of distributed synchronous collaborative software brainstorming and design meetings.

- To define a conceptual model that describes the cooperative activities of global software development meetings and study the application of the model during several cases studies, incrementally adapting and improving the model to cope with the issues encountered during the experiments.
- To propose a flexible object-oriented framework that makes possible synchronous distributed cooperative work over composite artifacts (UML artifacts augmented with knowledge creation, group awareness, group memory, Rationale, and history information). The flexibility of the framework consists of providing a reusable and extensible software architecture that can be deployed and configured according to the developers featured activity; such as distributed mind mapping, which needs different tools than those for UML design (e.g. a different toolbar).
- To introduce an experimental-based empirical approach to identify the requirements for a framework investiture to support distributed synchronous collaborative software design meetings; to investigate the implications of this kind of approach in an environment where the user-involvement is a criterion, to build frameworks within the related problem and solution domains.
- To refine the empirical approach to an innovative high-level design method outlining a process of designing software platforms for distributed software development meetings.

2. Scope

This dissertation addresses a research problem that embraces several fields: object-oriented software development, computer supported cooperative work, and human-computer interaction. As stated by Nierstrasz and Tsichritzis in [111], object-oriented software development has reached a relative maturity that enables software developers to have a shared understanding of an oriented software development process and a life cycle. Computer supported cooperative work (CSCW) on the other hand, is a multidisciplinary research area involving a number of fields including computer science, sociology, psychology and linguistics, although it does not have a universally accepted definition. According to Bannon [83], there are at least five distinct ways of viewing CSCW:

- CSCW as a loose agglomeration of cooperating and at times competing communities. This view sees CSCW as an umbrella term with little content other than something that has to do with people, computers and cooperation. Hence, CSCW can be seen as a forum where people from different disciplines and with partially overlapping concerns can discuss issues of mutual interests.
- CSCW as a paradigm shift. This focuses largely on the organization aspects and human factors that are taken into account when designing computer support systems, as opposed to the technology-focused design of computer support systems.
- CSCW as software for groups. This view sees CSCW as a research field focusing on people working with computers in groups. The term *groupware* is often used to name this area of CSCW.
- CSCW as technological support of cooperative work forms. Here the emphasis is on understanding cooperative work as a distinctive form of work, and on supporting these cooperative work forms with appropriate technology.
- CSCW as participative design. This view of CSCW is an alternative to traditional system design, where end users are involved more thoroughly in the design process.

CSCW and Software Engineering follow different strategies in organizing cooperative work. In software engineering, formal structure and methods are used to structure the software development process. CSCW, on the other hand, supports information management and cooperation within groups. CSCW supports cooperation in software engineering on two levels, the macro-level and the micro-level:

- Macro level: cooperation between different divisions or groups (technical and analysis divisions)
- Micro level: cooperation within a group (implementation, modeling etc.)

This dissertation shows how software engineering methods and techniques can support the micro-level. Vice-versa, the CSCW concepts are used to foster support to group software development in specific situations such as global

software development projects.

To cope with the distribution constraints, the CSCW researchers have suggested several topologies [34], such as the replication of data (decentralized), or sharing the data (centralized). In both situations, several disadvantages arise and can make the feasibility of global meetings difficult. This shows in particular, in distributed brainstorming and software design meetings, at which users create, update, and interact with complex and huge models, which may cause an important network traffic and a slower response-time.

Large and complex software models and artifacts cannot be replicated without causing a slower feedback of the system to the users [117, 37], who may think the system has crashed or is not able to handle their actions. That's why the response-time of the users interaction with the system may contribute to the success or failure as well (that is, the acceptance or not) of the real-time groupware systems. To provide good response-time, we suggest to use a centralized-like architecture in which no data are sent to the users apart from a notification to create the same data locally. That is, we replicate the users actions, which yields data locally if needed. A similar technique called *Active Replication* is used in distributed database replication [84]. Existing systems that replicate data among the distributed participants, leads to a substantial network traffic in sending all data back and forth between users.

From the experiments we conducted, we noticed an improvement of the response-time and the reduction of many interaction conflicts. As a result users share only a description of the model, which in return is displayed the same way to all of the users. Figure 1.1 shows a description of the notification flow and communication between a central repository, which contains the model, built by the different users, and the clients.

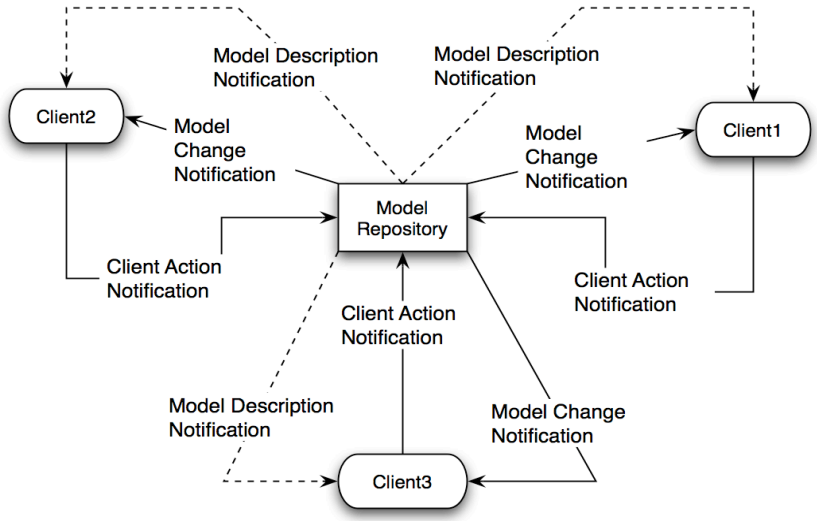


FIGURE 1.1. Dynamic Model of SCOOP Showing the Repository-Client Interaction

3. Problem and Solution Domains of the SCOOP Framework

SCOOP is a component-based framework [105]. Its main domain activity is software brainstorming and design in distributed synchronous meetings. SCOOP breaks down the domain activity into several cooperating components, such as *Activity, GroupAwareness, GroupMemory, Location, FloorControl, Communication, Rationale,* each with a responsibility. This way, components can be substituted with diverse implementations matching the context of the applications built on top of SCOOP. Examples of applications that are based on the same abstraction domain activity are GroupUML, GroupConceptMap, GroupBrainstorm, and GroupMindMap. These applications make possible the sharing and sketching of ideas, the identification of requirements, sharing use cases and scenarios between geographically dispersed users who can not meet in a single site.

Figure 1.2 describes the SCOOP framework, its components, and the

related applications built on top. Figure 1.3 shows an initial meta-model of SCOOP.

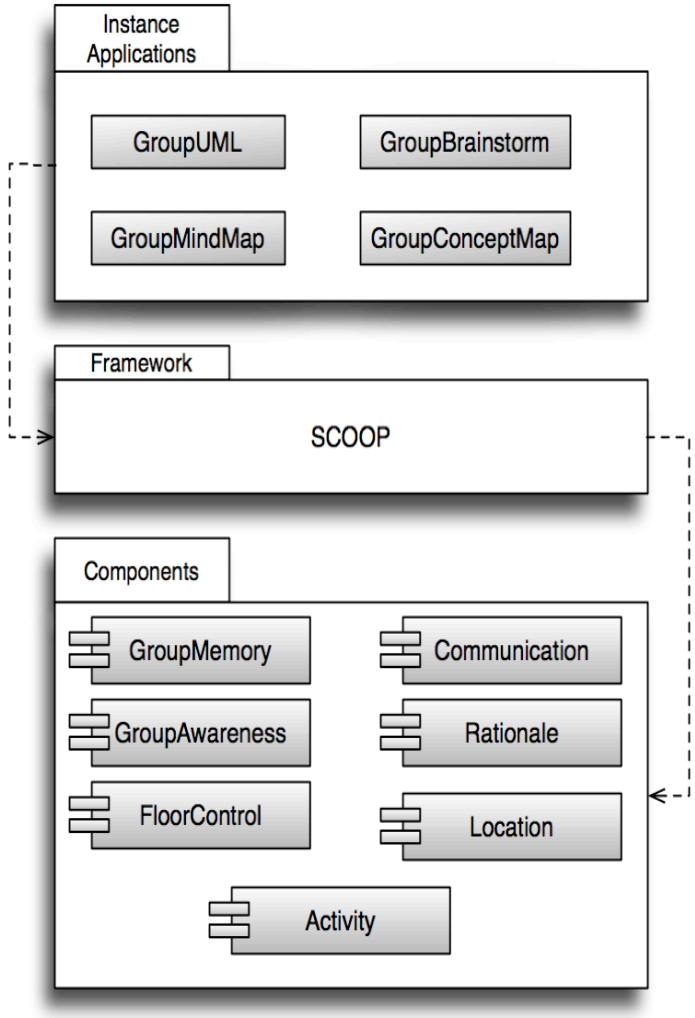


FIGURE 1.2. SCOOP framework is built on top of a set of loosely coupled components. Domain applications such as GroupUML are built on top of SCOOP.

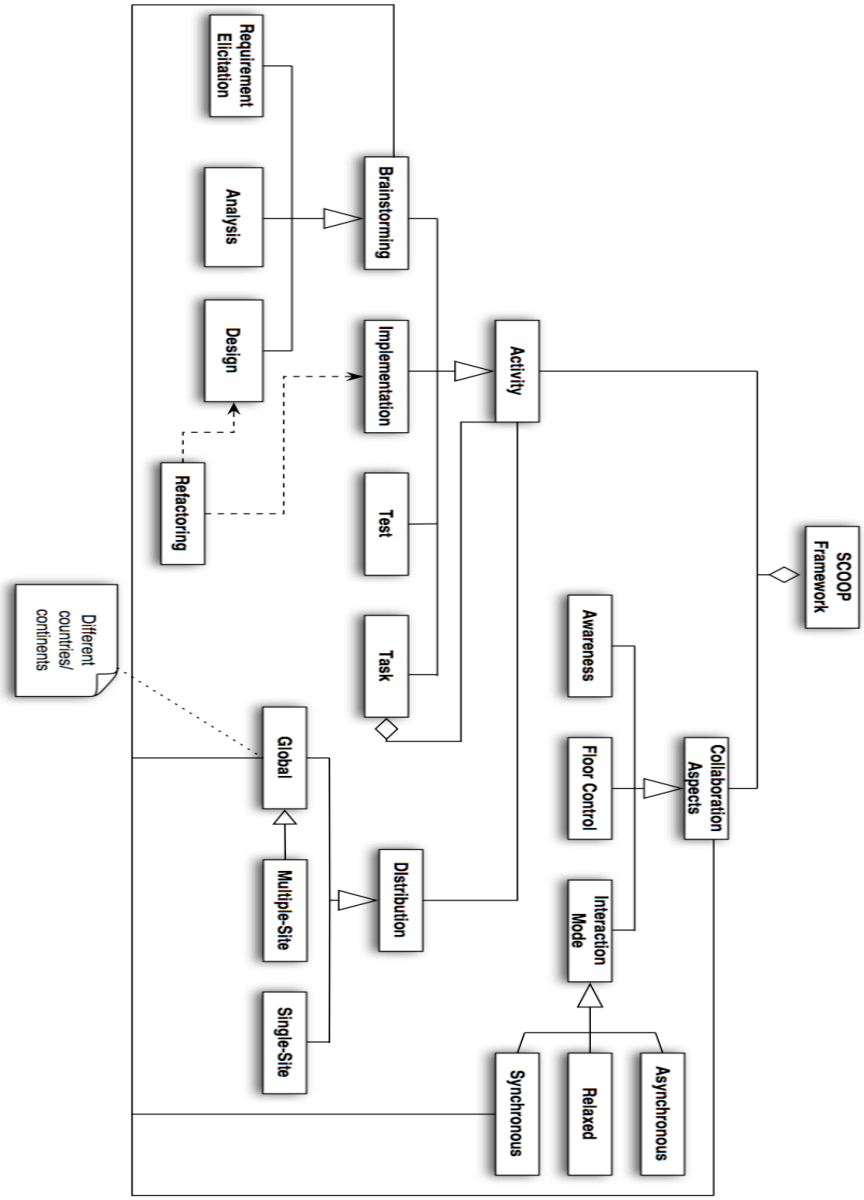


FIGURE 1.3. Initial Meta-Model of SCOOP

4. Approach

To deal with the issues described above, we conducted a series of exploratory case studies of distributed synchronous collaborative brainstorming meetings.

The formative experiments are conducted using SCOOP itself and served to the iterative exploration of requirements of SCOOP, and incrementally designing and developing new features of SCOOP. The new features were then used to support the subsequent experimental meetings. This cooperative design approach led to the development of an abstracting design method that uses the framework to develop and improve the framework itself.

Participative design elevates users from being objects of study to a role more intimately involved in the design process [22]. Cooperative design involves finding new ways for users to learn, participate, and cooperate with software designers. Since users play a central role in learning how a software system should operate, we adopted a participatory design methodology to include motivated users to codetermine how the framework will affect their activities in real life.

The first version of the SCOOP framework was initially developed using a feature-driven development process as shown in Figure 1.4.

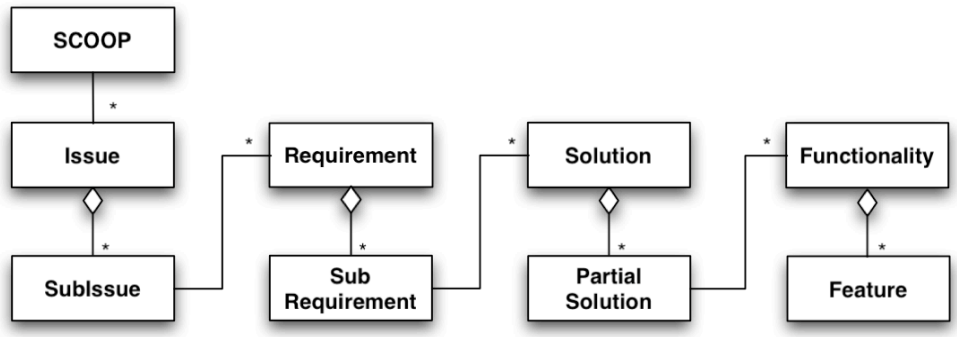


FIGURE 1.4. A meta-model of the feature-driven development approach followed in building a basic implementation of SCOOP

We built a groupware called GroupUML [106] providing such basic features as the sharing of visual graphic artifacts between dispersed users, used in the early stages of the identification of the requirements in building SCOOP. With GroupUML, it was possible to conduct distributed synchronous brainstorming meetings with senior computer science students in the Technische Universität München. Once the initial version was built, GroupUML was developed based on a participative design approach, by adding one feature at a time. Each feature was the outcome of one or more brainstorming and design meetings with users. Users spread over different locations and cooperated in brainstorming the requirements and designing the features. Users used the current version of SCOOP to implement and design the next version. This repetitive and incremental bootstrapping design cycle where an initial partial solution (version 1) is used to build a partial solution (version 2) is shown in figure 1.5.

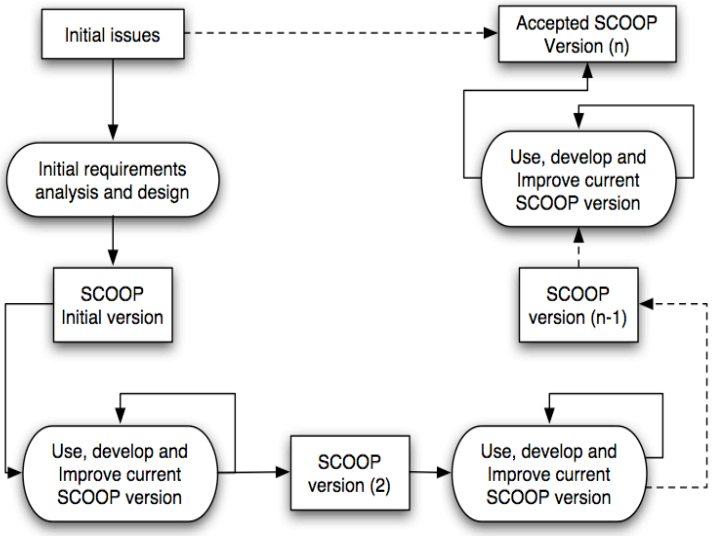


FIGURE 1.5. Initial Incremental design method used to develop SCOOP

We have termed this process as the *BID* approach [108]; described as follows: having initially an issue, we start by building a trivial and simple partial solution. In our case, the initial partial solution was built upon a user-centered design paradigm. The next step consists of using the partial solution to identify current issues and try to identify a possible solution to each. The possible solution is then built on one available solution from the previous cycle. The process takes as many cycles as necessary until it reaches a level of user acceptance.

This also means that all newly identified issues and faced problems (resulting from a regression, for example or a conflict between previous and current partial solutions) could be solved or workrooms have been found without compromising any functional requirement or functionality of the system.

5. Contributions

The core contribution of this dissertation is a flexible component-based framework support called SCOOP [107], for real-time distributed group software brainstorming and design meetings. The second contribution is a reference implementation and evaluation of a groupware called GroupUML[105]. GroupUML enables the sharing of graphical artifacts in real-time and capturing history and rationale knowledge. The artifacts are UML models augmented with rationale knowledge, group memory, history, and awareness information that supports concurrent management. GroupUML was used to explore several distributed software brainstorming and design meetings.

A third contribution is the design and conduction of a series of case studies as a proof-of-concept of the research hypothesis, that distributed real-time software design and brainstorming meetings (same-time/different-place) are feasible. A fourth contribution is the design of a conceptual framework process for incremental and self-improving tool development.

6. Thesis Structure

The structure of the thesis is described as follows:

In chapter 2, we introduce various definitions and terms used in the dissertation. We discuss global and distributed software development projects, groupwork and collaborative software brainstorming and design meetings. We introduce distributed model-driven development meetings.

In chapter 3, we describe the application domain. We introduce global and distributed software development activities and formal meetings. We identify issues of distributed meetings, the problem domain and the solutions envisioned. We explain why supporting software design activity in distributed software development projects is important.

In chapter 4, we describe the requirement for the SCOOP framework, then, we describe distributed group meetings activities for software brainstorming and design. In particular, we elaborate a conceptual model of brainstorming and design-based activity life cycle as the basis for developing and improving the SCOOP framework.

In chapter 5, we provide a detailed description of SCOOP as a contract-based component framework and the reference implementation.

In chapter 6, we describe three sets of case studies and the experimental settings and context, finally we discuss the results.

In chapter 7, we discuss and describe our reflection on the particular design process we used and extended it into an agile approach bootstrapping in the design and development of frameworks. We start our discussion by an exploration of the approach using SCOOP to improve the design of SCOOP. We generalize the bootstrapping concept in groupware development.

In chapter 8, we conclude the dissertation and we outline future directions that can be explored based on the results of this research and summarizes the implications of the results for distributed and global software development.

CHAPTER 2

Definitions and Terminology

2.1.Introduction

Globalization has pushed software development into a new and more difficult dimension. Large software systems are no longer developed on a single site, but are rather developed across sites, cities, countries, and even continents. Thus, new aspects of collaboration, communication, and coordination complexity have been introduced. Hence, the complexity of developing software systems has increased.

In the next section we will define and outline our main concepts of distributed software engineering, and in particular distributed software development.

2.2.Distributed Software Development

The term "distributed software engineering" is ambiguous. In an attempt to cover the research aspects of distributed software engineering, Kramer [62] provided the following description: "Distributed software engineering includes both the engineering of distributed software, possibly local, and the process of distrib-

uted development of software, such as cooperative work". The former deals with software products that are destined to be deployed in a distributed environment (distributed computers, distributed databases, networks, etc.). The latter deals with the process of distributing the activities of software development within a scattered groups of developers.

Although the two definitions seem independent, they are related. In the first definition, although the concern is about developing software deployed in a distributed environment, it can be developed according to the second definition, that is, within a geographically distributed development team. Vice-versa, the software product that is developed according to the second definition may also be destined to support distributed deployment.

We can, therefore, distinguish two definitions depending on whether the concern is on software product engineering or the process of software development.

Definition 1.1. *Distributed software engineering* is a field that covers the issues of software products that are deployed in a distributed environment.

Definition 1.2. *Distributed software development* is the process of distributing (concurrent) collaborative software development activities within a geographically dispersed group of developers.

This dissertation is concerned with distributed software development as described in definition 1.2. It focuses on investigating the special case of distributed collaborative software brainstorming and design, identifies and addresses the related issues. It presents solutions towards resolving these issues.

Several factors are contributing to the dissemination of distributed development projects, technological, technical, economical, organizational, and social as well. The increasing quality and availability of internet-based technol-

ogies (such as high-speed ATM-based platforms), makes distributed software development a more viable option for many organizations. The improvement of technological support in communication infrastructure (e.g. groupware tools, video conferencing), the development of internet technologies, bandwidth and performance, have dramatically changed the way software is developed. Another reason for distributing software development is the availability of technical resources such as specialized hardware, which may only be available at certain locations, for example confidential prototypes or supercomputer platforms which are very expensive to replicate. Thus, many projects have to be developed by geographically distributed teams.

For economic or organizational reasons, projects cannot always be limited to one company or a single location. For instance, time to market needs to be reduced in the internet-time: business success is much more strongly determined by being the first to sell and market a new technology [54].

Moreover, strategic partnerships, joint ventures, and global companies share a need in supporting distributed software development efforts. Companies can develop software products, which are then promoted by other partner companies. Existing products may need to be customized and supported by companies other than the original developers. Global companies with offices in many countries may conduct joint product development among divisions.

French [44] studied five commercial software development and maintenance projects to identify the advantages and disadvantages of the distributed work model. The following organizational factors contributing to the distribution of software development efforts were identified:

- Client may request or require on-site support. It may be necessary to have some project members located on or near a client site.
- Project members are unwilling or unable to travel or relocate to other sites.
- Skilled workers necessary to the success of a project are based at different sites.
- Travel or relocation costs for moving a large number of project members to a different site could be exorbitant.
- Technical resources such as specialized hardware are only available at cer-

tain locations.

- Organizations feel uneasy about having staff from another company work at its site, or the software development organization did not want to have its staff working at a customer site.
- There is a shortage of office space at a given location.

In spite of the many reasons requiring a distributed development approach, distribution alone does not automatically represent a gain in productivity and quality. For instance, Microsoft considers developing new products on site as an advantage because whenever problems occur, caused by work interdependencies, developers can efficiently discuss and solve them in informal face-to-face meetings [93, 137], which presents numerous advantages as stated in [44].

However, several advantages in distributed work have been identified as well. The enhancement of organizational productivity, delivery of customer services, reduction of traveling time are frequently mentioned benefits. In [58] E-mail is noted to be an important technology supporting telework. Broadcasting capability, management of communications, access to information resources, low communication cost, file transfers, and temporal and spatial flexibility are all communication benefits original to E-mail and the web which helps to make distributed work possible. In [112], a software design experiment was conducted to compare the quality and creativity of system designs for groups with two different meeting environments. Forty-one groups of about five participants each took part in the study. One set of groups held traditional face-to-face meetings, while the other set used distributed asynchronous computer conferencing meetings. The study showed that the distributed asynchronous groups produced more creative designs than the designs produced by the face- to-face groups. Another observed outcome was that the quality of the designs produced by the distributed asynchronous groups were higher in quality than face-to-face groups. This research suggests that the more creative and higher quality results of the distributed groups may be attributed to several factors. The use of a collaborative writing tool, the ability to work in parallel to combine ideas, the pro-

duction of a written memory reducing the number of lost ideas, and the increased connectivity using computer support are all factors that may have led to better results from the distributed groups in the study. Similar findings are described also in [4].

Different sites can be allocated to different functions. For example, separating testing from development is an interesting opportunity for having solid tests: the less testing is influenced by development, the more independent test scenarios will be, giving a greater chance of running into errors that many developers have ignored or missed. Distributed work requires more discipline to keep track of documents and deliverable. Processes have to be well defined and understood, which leads to a more stable project environment [44]. Since most of the communication and collaboration within distributed teams is supported by computer and video conferencing tools, a larger part of the project history and the underlying artifacts can be made explicit and recorded. Such a record can be (re)used by the project participants to document and justify decisions better and by the company to accumulate corporate knowledge [7]. Staff from a broader pool of skills can be assigned to virtually any project.

Several issues and difficulties are resulting from distributing software development over multiple sites. In the following we discuss some of these.

In [55, 56], an embedded system product at Lucent Technologies that is described, was developed in two locations: Germany and the UK. This project revealed many problems resulting from the distributed development process. Unit testing and development was disturbed by incomplete specifications. Implementation diverged from the design document, and the design document was not updated. Bug reports were generated from out-of-date design documents that identified ill-functioning components. While the components were working as implemented, the tests generated at other sites used design documentation that was never updated after designs changed. Many of the coordination problems observed in this project can be attributed to communication losses incurred because of distributed development. These are the same communication losses

that can occur with all distributed work. In [44], French has found that on the one hand, project members tend to be less committed to a project when a lot of the communication takes place by E-mail, that is in an asynchronous way. On the other hand, sites can communicate too much and spend more time communicating than doing useful work [44]. Collaboration breakdown happens also when distributed groups do not know whom to contact about particular issues because of the difficulty of making contacts.

As a result, communication and coordination issues complicate distributed work and have to be dealt with when designing groupware supporting distributed teamwork. Synchronous groupware makes communication and coordination issues and their corresponding solutions explicit.

Several technical issues and difficulties result from distributed software development. We describe in the following the relevance to this research.

One of the difficulties in distributed software development is that the range of issues to be addressed cover many disciplines, including CSCW (Computer Supported Cooperative Work), process improvement, software configuration management, project management, organizational theory. Moreover, there are few experience reports in the literature on distributed development.

While there have been considerable recent advances in CSCW technology [32], the advancement of framework support for collaborative distributed software development has been neglected.

Several issues have to be considered in developing groupware support to distributed software development, among these we cite the most important ones related to our research:

Meetings. This area includes the issues related to making distributed synchronous meeting possible, in particular those that are intended to support collaborative brainstorming and software design.

Collaboration. This area includes the issues related to supporting the collaboration of different individuals and teams over distance, time and communities. This includes issues of increasing awareness, supporting communication,

and supporting negotiation.

Artifact management. This area includes the issues related to supporting the exchange and tracking of different work products at different stage of development. Work products include system related artifacts, such as specifications, designs, code, test scripts, as well as work-related artifacts, such as project plans, status reports and process descriptions.

Project management. This area includes the issues related to planning, coordinating, tracking, and controlling different parts of a project. In particular, workflow management and process modeling fall into this area.

Knowledge management. This area includes the issues related to managing the knowledge assets of an organization, including experts, lessons learned documents, artifact templates, best practices and process improvement.

Localization. This area includes the issues related to customizing artifacts, work, and tools to a specific site to support its local needs better while enabling the site to remain part of the global organization.

In the following, we will focus on the role of meetings in distributed software development, in particular the issues related to distributed synchronous collaborative software development meetings.

2.3.Distributed Collaborative Software Development Meetings

Meetings are the most important vehicle for human communication. They are so common and pervasive in organizations, that many take them for granted and forget that, unless properly planned and executed, meetings can be a terrible waste of precious resources.

There are two types of meetings, formal meetings and informal meetings. Formal meeting have agendas known in advance to all participants. The meeting has a specific agenda and take place at a predefined location considering time constraints of the participants. The participant's relationship results from social

aspects, their company's culture, their hierarchical structure, their affiliations.

Informal meetings are meetings that have very few or no constraints at all upon meeting location, participants, and subjects of discussion. There are two types of informal meetings:

Casual meetings 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 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 the meeting is off. 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.

Software developments organizations schedule meetings as one of the most important frequent activities. Meeting participants may undertake long trips for attending a meeting. The activity undertaken within a meeting is one of the characteristics that define the type of the meeting. Discussion meetings deal with discussing issues in a given context of the meeting. Software development meetings are specially dedicated to software development activities.

Definition. *Distributed meetings* are meetings where participants are geographically dispersed and use a computer and networking infrastructure for communicating.

We extend the previous definition to define distributed software development meetings.

Definition. *Distributed software development meetings* are distributed meetings where participants conduct software development activities.

Definition. *Brainstorming* is an organized approach for producing ideas by letting the mind think without interruption. Brainstorming can be done either individually or in a group; in group brainstorming sessions, the participants are encouraged, and often expected, to share their ideas with one another as soon as they are generated. The key to brainstorming is not to interrupt the thought process. As ideas come to the mind, they are captured and stimulate the development of better ideas. Brainstorming has some limited use in enhancing creativity in that generating a broad selection of ideas may lead to a unique and improved concept.

Brainstorming is concerned with more than idea generation. Osborn [113] considered idea evaluation to be an essential element. The rules he came up with are the following: no criticism of ideas, go for large quantities of ideas, build on each others ideas, encourage wild and exaggerated ideas, and evaluate and select relevant ideas.

The brainstorming activity is composed of three sub-activities.

The *Fact finding activity* contains two sub-activities: problem definition, and preparation. First, we define the problem to solve, where we may need to break it down into smaller problems. Next, we gather any information that might relate to the problem.

In the *Idea generation activity* we generate and find ideas without any constraints.

In the *Solution finding activity* we evaluate and select the best ideas among the ones generated.

Brainstorming works well for problems that have many possible answers such as design. In exploratory design meetings, we often iteratively jump from the *solution finding activity* to the *fact finding activity* till we meet an acceptance criterion to the underlined problem like described in Figure 2.1.

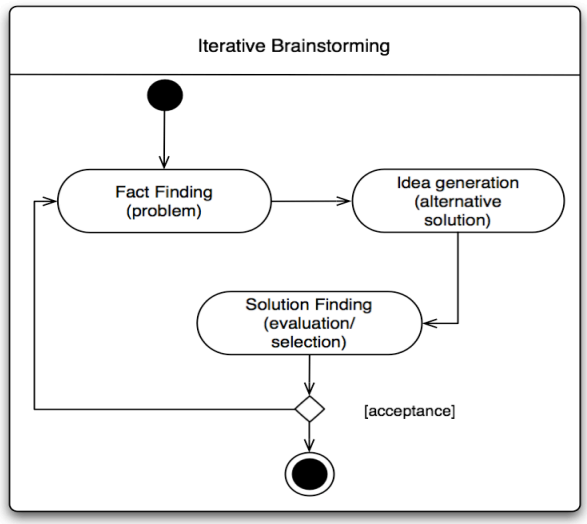


Figure 2. 1: Iterative brainstorming activities (UML Activity Diagram)

Based on this definition we define object-oriented brainstorming activity as follows:

Definition. *Object-Oriented Brainstorming*: is the act of brainstorming within object-oriented analysis and design activities. It is concerned with finding objects, components, systems, evaluate findings and consolidate ideas.

2.3.1 Group Memory of Meetings

Group memory represents the knowledge produced during group meetings. It takes two forms, short-term memory and long-term memory.

Both are crucial to run meetings and to enable long-term collaborative support within projects where necessary information are stored to describe artifacts, their history, the rationale behind them, and the users who created them.

Short-Term Memory. It has a short life-time and is useful only during the meeting itself. It helps individuals to contribute to the meeting the right way and efficiently. Short-term memory constitutes the information such as what is said, and the events happening during the meeting which is crucial for the continuation of the meeting. Meeting attendees are usually taking notes for themselves and each is acting as a self-minute taker. The result is that each one has his own description of facts happening during the meeting, and each has his or her own interpretation of what happened and what was decided. This leads to some conflicts in the group. To cope with these issues, Michael Doyle & David Straus in their book "How to Make Meetings Work", presented a trivial solution which consist in to adopt a collective short-term memory.

Long-term memory. In formal meetings, the minutes as recorded by the minute taker constitutes the long-term memory of the group.

Collective short and long-term memory. Distributed software developers in particular, need to deal with both short-term memory as well as long-term memory to coordinate their activities during and after distributed meetings. Short-term memory can be individual so every one has his own description of what he or she is going to say or what has been short said or decided. Long-term group memory is generally recorded by the minute taker and will not be available until the next meeting.

Collective or shared short-term memory can be used to cope with the different individual recordings. It consists in writing instantly all short-term mem-

ory informations in front of the group, using a white or a black board. There several benefits from using this technique:

- Individuals are no more overwhelmed with information and figures and charts that are brought up.
- Concentrating on own ideas is no more necessary since this can be shown on the collective memory and one is open to the group's new ideas.

Distributed Group memory. Such a concept is not known yet. In our view, distributed group memory is the extension of the single site (same-time/ same-place) group memory into a shared distributed memory in distributed settings. In the following we provide its definition:

Definition. *Distributed Group Memory* is the collective or shared memory of a physically distributed group of individuals.

Several issues arise from the definition we provide. For instance, how to design and represent distributed group memory, how to integrate it into a meeting system, what is the impact using it in distributed group meetings, what kind of enabling technologies that can support implementing distributed group memory. These questions are answered along the thesis.

2.3.2 Enabling technologies for software development meetings

Recent developments in networking and multimedia technologies are promising for the wide dissemination and use of video conferencing technology and services in software development meetings [28].

Electronic meeting systems (EMS) are communication means including teleconferencing, computer conferencing, and group decision support systems, which utilize information technology to support distributed meetings. These systems support electronic brainstorming as well as various decision making processes. Electronic meeting systems aim to make group meetings more productive by applying information technology. EMS technology is designed to

directly impact and change the behavior of groups to improve group effectiveness, efficiency, and satisfaction of the users with the outcome of the meeting.

Electronic meeting systems are used for meetings that requires the coordination of several people. As long as the activities are of general nature or simple ones, meetings can be conducted successfully. Complex tasks need more preparation and an effective planification to conduct successful meetings. The goal has been to understand how to build computer tools to make such meetings more effective.

In [109] it is argued that EMS are not confined to supporting only meetings that occur in the same place at the same time. Using modem information technology, people can come together for a meeting even though they are separated in time or space. There are a few published reports of rapid and wide adoption and diffusion of this technology. The most deployed class are Group Support System (GSS) technologies, include products such as Lotus Notes and Microsoft NetMeeting. However these systems were little or not used within distributed software development organizations [12].

For distributed software brainstorming and design meetings we need to develop and use more dedicated software infrastructure to support the associated activities. For example object-oriented software design requires support of UML tools as well as for rationale knowledge accumulation and management.

2.3.3 Global software development meetings: levels of distribution

Global and distributed software development activities can occur at different places and at the same time where synchronous software developing meetings can take place. It can also take place in different time where asynchronous activities can be conducted. We can not talk about ‘asynchronous meetings’ since meetings suppose that people come together wether in the same place or different places.

Researchers distinguish between three dimensions where distributed software development meetings become more difficult:

• **Geographical distribution .** In CSCW, this dimension is typically referred as "same place vs. different place." The distances between the sites and the number of different sites impact on the difficulty for the sites to communicate and exchange information. This is because increasing distance between sites may lead to a reduced face-to-face meetings. Consequently, electronic means, such EMS, have to be used for communication and coordination between sites.

Moreover, mobile communications and the mobility of individuals and groups have introduced another dimension to the time-place Matrix [51, 14].

• **Temporal distribution .** In CSCW, this dimension is typically referred as "same time vs. different time." If the working schedules of different sites overlap, synchronous tools (e.g., phone, video conference, and synchronous groupware) can be envisioned for supporting collaboration. Otherwise, asynchronous tools (e.g., E-mail, fax, and asynchronous groupware) must be used, which result in a much higher latency in communication and decision making.

• **Community distribution .** Groups from different organizational cultures work together on a single project. Each group may have developed its own terminology, tools, and methods, making it difficult for them to coordinate their work, even if they are within driving distance or within the same time zone. In the case the different groups are forced together (e.g., as a result of a merger), cultural clashes may degrade the communication to the point of project failure.

2.3.4 Distributed synchronous collaborative software modeling meetings

When working hours overlap, distributed developers can conduct synchronous collaborative brainstorming meetings as defined. Developers can not only collaborate asynchronously on activities independently from each other (e.g. coding, testing, integration) but also can collaborate synchronously on activities such as brainstorming and designing architectural models.

2.3.5 Importance of software models

Models help us understand a complex problem and its potential solutions through abstraction. However, Selic in [16] states that for historical reasons, models often play a secondary role in software development. Nevertheless, models merit better consideration as the potential benefits of using models are significantly greater in software development than in any other engineering discipline [16]. Models are more than visual diagrams. Models encapsulate a wealthy bunch of information about brainstorming, analysis, specification, and functionality of software design issues. Models should not end up as documentation like in most traditional software development projects. Documentation easily diverges from reality because it's difficult to maintain it synchronized with requirements or code that change frequently.

The major advantage in focusing on models instead of implementation is that models are expressed using concepts that are much less tied to the underlying implementation technology and are much closer to the problem domain than the most used programming languages. This makes the models easier to specify, understand, and maintain. Selic claims that in some cases, it might even be possible for domain experts rather than computer professionals to produce systems through models.

In the next section we present model-driven development concepts and its relation to this thesis.

2.4.Model- Driven Development

Model-driven development (MDD) is an OMG initiative to develop standards based on the idea that modeling is a better foundation for developing and maintaining systems [135].The aim of Model-driven development is to drive the entire software engineering process through models and to reduce the importance of code, and focus on building applications reliably and according to requirements. Model-driven development is part of a much broader concept

called Model-Driven Architecture (MDA). MDA represents a conceptual framework for an approach to model-driven development. Models are constructed, viewed, developed, and manipulated via UML in a standard way at analysis and design time. UML models allow an application design to be evaluated, reviewed, and modified before it is coded, when changes are easier and less expensive to make. The Meta-Object Facility (MOF) standardizes a facility for managing models in a repository. XML Metadata Interchange (XMI) is an interchange format for models, based on the MOF and the language XML.

2.4.1 Model-Driven Development goals and benefits

The goal of MDD is to have as much as 90% of an application to be automatically generated from UML diagrams [60]. Specifications, implementation, building, debugging and testing can all be done at the model level.The important message of model-driven development is that it allows developers to focus on solving the design challenges and adding new functionality, instead of spending most of their time fixing syntax errors, stopping memory leaks or over low level bugs.

The underlying motivation for MDD is to improve productivity, that is, to increase the return (*ROI*) a company derives from its software development effort.

The benefits of model-driven development are significant to business leaders and developers as well.

Integrated support environments to MDD (that is, environments that allow the development of domain-models, code generation from these models, and the generation of models from code through reverse-engineering) are still not available and the very few that exist are not mature [64]. To date, most commercial tool support for MDD focus their effort on automating code production from visual models. However, a number of researcher have developed tools (e.g. ArgoUML, Eclipse Modeling Framework, AndroMDA) in an attempt to enable ‘round-trip’ engineering where users can switch seamlessly between a UML model and corresponding implementation source code. Several researcher even

argue that from models we can produce good enough code that can be maintained; but generating models from code can not equal the quality and the abstraction level of the original models designed by software developers.

MDD researchers are focusing more on tool support and are neglecting the underlying activities of development. In particular there is no support to model-driven software development for distributed and multi-team developments. This thesis provides a first step towards a support to distributed Model-Driven Development activities.

To support distributed Model-Driven Development meetings we focus on object-oriented brainstorming, analysis, and design meetings using UML in distributed settings. In particular, we focus on the collaborative creation and manipulation of graphical UML artifacts within distributed development meetings. The artifacts are used as a vehicle of communication among the various stakeholders, and helps bridge the complexity of the system.

GroupUML is a research prototype that focuses on UML modeling, it does not support code generation nor round-trip engineering. It is intended to investigate the initial requirements for distributed support to UML modeling and concurrent management of visual artifacts. Systems developed with GroupUML are not made up just from the UML models, but also from other artifacts. This includes requirements, rough sketches of the system, rationale, and business cases.

CHAPTER 3

Problem Statement

We distinguish two different types of global software development process models: Horizontal process models and vertical process models. Software process models are abstract representations of software processes. A software process is a set of activities that are performed by the project participants toward a specific goal and the dependencies between activities. In the following we describe the two different process models.

In the horizontal process model (see Figure 3.1), the distributed software development process refers to the distribution of independent or loosely coupled activities that are spread over different locations. That is, the process consists of loosely coupled activities that can be performed independently from each other. For example, the design of a system architecture, the implementation, and the testing activities could be conducted independently in different sites by different development groups. Every single site is fully responsible for its deliverable. This type of process model is adequate for different sites that are located in different time-zones where communication an collaboration take place asynchronously (see Figure 3.2, dashed ellipse I).

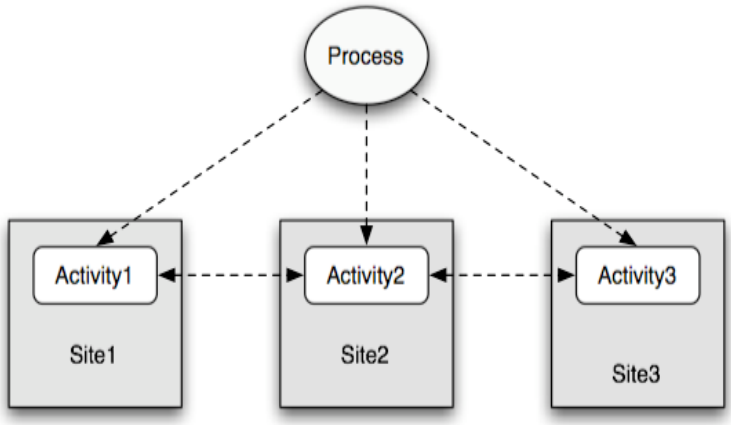


Figure 3. 1: Horizontal process model: distributing the process of development over multiple sites

In the vertical process model, each activity of the development process is simultaneously shared and performed synchronously by different groups of developers located at geographically distributed locations (see Figure 3.3). As a result, brainstorming and software design, coding, debugging, inspecting, and documenting can be jointly performed by several participants in the same time. Sites that share the same time-zone and being in different locations can perform concurrent synchronous work (see Figure 3.2, dashed ellipse II).

Sites that are in different time-zones (see Figure 3.2, dashed ellipse I) can perform synchronous work only when scheduling special meetings where one site compromises in attending meetings very late or very early with respect to their local time.

The vertical process model with focus on concurrent software brainstorming and design in global software development projects, provides the context for this dissertation

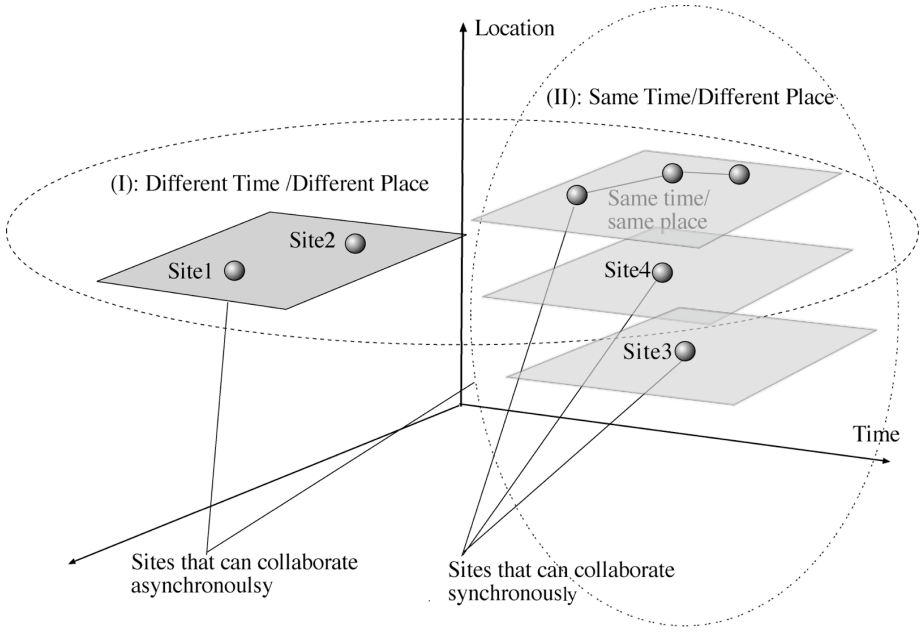


Figure 3. 2: Collaboration dimensions of multiple-site development projects. The dashed ellipse (I) shows the Horizontal process model, the dashed ellipse (II) shows the Vertical process model.

Alcatel is an example of companies that use the horizontal process [21]. Alcatel Telecom is a global telecommunications supplier with a large number of research and development projects that typically involve several countries around the world. Alcatel Data Networks (ADN) is the business division for switching and routing line of products. Software development in this business division is handled in a central R&D group of several thousand of multifarious software engineers who are distributed throughout the globe in more than 20 development centers in Europe, the US, North Africa, Asia, and Australia.

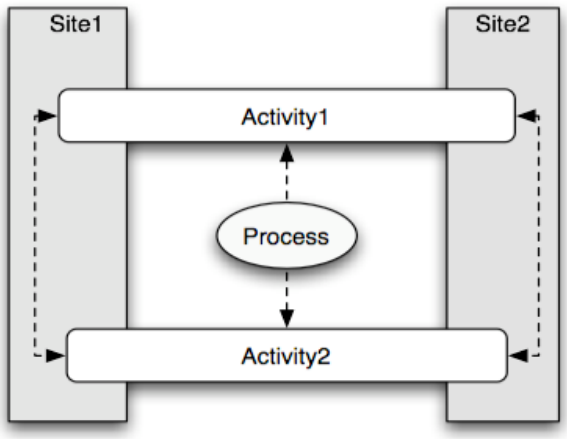


Figure 3. 3: Vertical process model of distributed software development

Alcatel distributes its software development process over different continents. Software systems are designed and developed in one site, tested and integrated in another site, deployed to the client at third site. Several travels on site are scheduled to solve issues. Problems that were not discovered and solved during the development are fixed on the client-site. This necessitates additional mobilization of many people that are involved in the project.

ADN Clients (usually a government, group of Banks, telecommunication, and space companies) have the opportunity to get a specific-software-system that matches their needs. Usually new clients visit the main (Management) site, where a running demo platform composed of a hardware platform (Alcatel PSX switches, Alcatel ACX concentrators, 10 to 20 SUN workstations and NCD/ Tektroniks Terminals) and a distributed peer-to-peer software platform (Alcatel 1100 Network Management Unit NMU, VPN Graphical Station VGS, Network Management System NMS). The demo platform is a complete replication of the development site platform. The client checks wether a customization is needed and specifies additional client-specific-functionalities through RFCs (Request

For Change) that are communicated to the executive site. The negotiation of new or changed requirements happens at both the executive site and the development site.

The executive site coordinates all the different sites and serve as an official relay between all parties. In this site, all the major interactions with the client take place. In practice, all parties are encouraged to directly communicate with each other and with the client.

The development site is involved in all the activities of negotiating requirements, designing and developing the software. As a result here an intensive interaction and collaboration takes place with the clients. Several predictable travels to client-sites are scheduled each year. Often the development site acts on behalf of the executive site, where software development team leaders share decisions about software deliverable and deal with RFCs and clients requests. Additional interactions with the system and integration test site happens each time there is a client-software release.

The role of system & integration test site ensures that the system fulfills the quality requirements defined by the executive site and the client requirements. Regression tests are performed by the team in this site. The development site and the system and integration site have platforms that are a replication of the client platform to lessen compatibility problems and to discover hidden bugs.

Fault reports describe bugs and missing functionalities.They are the communication means between the development site and the system & integration test site.

Collaboration between the sites happens through E-mails, audio-conferencing and if necessary, traveling and meetings. Often travel has to be scheduled in a short time frame. Because of the high cost of travels, whole team of developers is often sent to the client-site to pursue development and bug-fixes for a period of time. Figure 3.4 outlines the collaboration process between involved sites in the parallel model used by ADN.

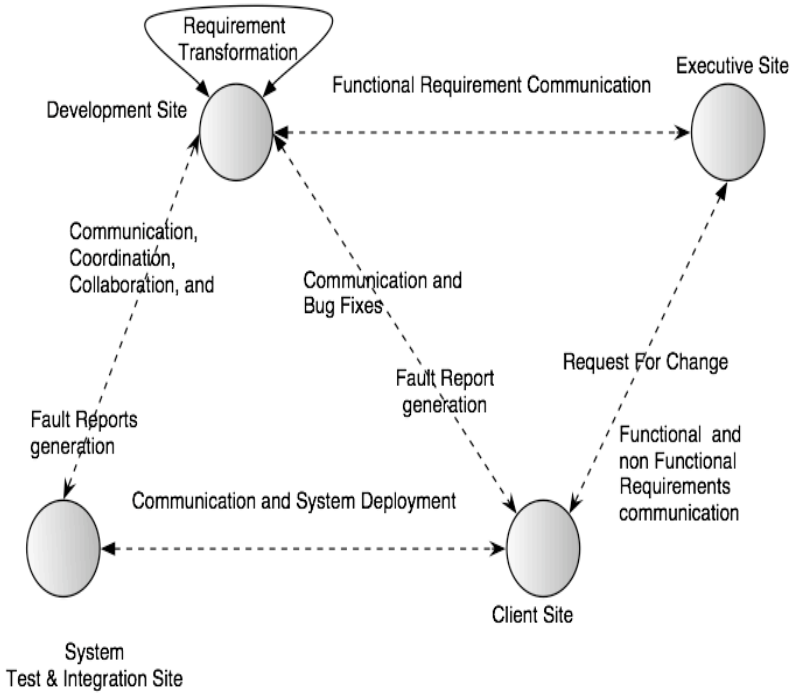


Figure 3. 4: The Process of Distributed Development of Software Within Alcatel Data Networks

3.1.Global Software Development Issues and Challenges

In the following we outline the major issues faced by researchers in global distributed software developments.

As developers and stakeholders have started to collaborate in projects across global distances similar to the process described for Alcatel, new challenges have emerged [118]. Many researchers have pointed out many issues that surfaced in their research on distributed, remote collaboration. For example less

frequent and predictable collaboration, and the loss of social contact.

In [91], Meadows describes several challenges that make managing requirements analysis process with remote sites difficult. He found that connecting people across sites, including channeling of communications is complex, and dividing work, and managing interdependencies in global projects is difficult [92].

Millar in [94] states that on-site visits and liaisons play a crucial role. On-site visits, in particular, help to lessen the difficulties resulting from getting deliverable from people at counterpart sites. Another issue resulting from distributed development is the lack of informal, direct communications that are more common in collocated work settings. This leads to lack of understanding of counterpart's context as described by Cramton in [27]. For successful communication using electronic media, new skills and attitudes are required [67]. Distributing the work processes result in delays that are described by Jarvenpaa & Leidner in [68]. Finally Majchrzak in [88] emphasizes on the role of groupware in solving issues in distributed development projects.

Exploratory studies have investigated geographically distributed collaboration as a new scientific research area. However, these studies have not addressed areas of synchronous joint software development, in particular joint brainstorming and software design. The literature shows that only very few experiences are conducted in supporting synchronous software development activities. For instance, Damian [29] reported on the use of an industrial groupware tool (TeamWave) for the requirements engineering phase. The tool has been tailored to support collaborative negotiation of requirements between geographically separated agents. The requirements engineering process can be structured via customized virtual rooms while the discussion of multiple perspectives is facilitated by specific TeamWave tools.; L. Brothers presents ICICLE ("Intelligent Code Inspection Environment in a C Language Environment") a software system intended to support distributed asynchronous code inspection by augmenting the process of formal code inspection. It offers assistance in a number of activities,

including knowledge-based analysis and annotations of source code, and computer supported cooperative discussion and finalization of inspectors' comments during inspection meetings; distributed code inspection [18], Gaoyan et al. present Codebugger a cooperative debugging tool for Java that supports distributed group developers to participate in one debugging session and cooperate on solving problems at the same time [46].

Although it is not explicitly stated, most of the above described studies of global and distributed collaborative software development occur in the context of distributed meetings, during which large amount of implicit as well as explicit knowledge is exchange. Making tacit knowledge explicit, such as design rationale, helps to maintain consistency during meetings and to deal with change during the project life.

In the following we describe the issues and challenges of rationale in distributed meetings.

3.2.Design rationale challenges in distributed meetings

Design rationale is the justification behind the design decisions. It may include the assumption made about the system, the alternative considered, and the reasoning that led to these alternatives. Rationale information may also include the problems developers encountered, the options they investigated, and the criteria they selected to evaluate options.

Rationale can serve three different purposes:

Improve the quality of decision. by making explicit the main decision making elements, rationale improves the quality of decision making by systematically clarifying the possible options and their evaluation against well-defined criteria.

Help deal with the complexity of systems. by capturing rationale, the dependencies among decisions spanning multiple aspects of the system are made explicit, and the structure of the argumentation process is made explicit and visible in the system as well, thus developers ca better deal with the com-

plexity of the system.

Support efficient evolution of systems. with emphasis on system evolution, design rationale is particularly important. To support evolving a system efficiently, as much as possible of the design should be reuses. Immediate access to the design rationale can help in this process by following designers to review decisions made during the original design in the light of the changed requirements. They can then make decisions about wether parts of the system should be adapted or completely redeveloped.

The potential benefits of dealing with rationale capture within meetings are manifold [18]; In the short term, it could give designers a better memory of their meetings by exposing previous discussions points, and in the long term, it could help system evolution by the system maintainers through the access and the understanding of the system history and reasoning [94].

In distributed meetings, dealing with rationale capture and management is difficult and significant challenges arise. Rationale information is spread over different locations and users. Representation of rationale information has to be efficient and easy to access and manage as in single site meeting. It must support concurrent and distributed authoring.

3.3. Approach

Our main interest in this dissertation is to investigate the issues in supporting distributed synchronous model-driven development meetings of geographically distributed developers and stakeholders according to the vertical process model. Second, to identify the requirements for a system that supports synchronous meetings involving object-oriented brainstorming, analysis and design and the accumulated rationale knowledge. Third, to build an extensible and adaptable framework that serve as a reference model.

To provide flexible software framework support for distributed software design meetings, we used design process of the framework for the thesis outlined in figure 3.5.

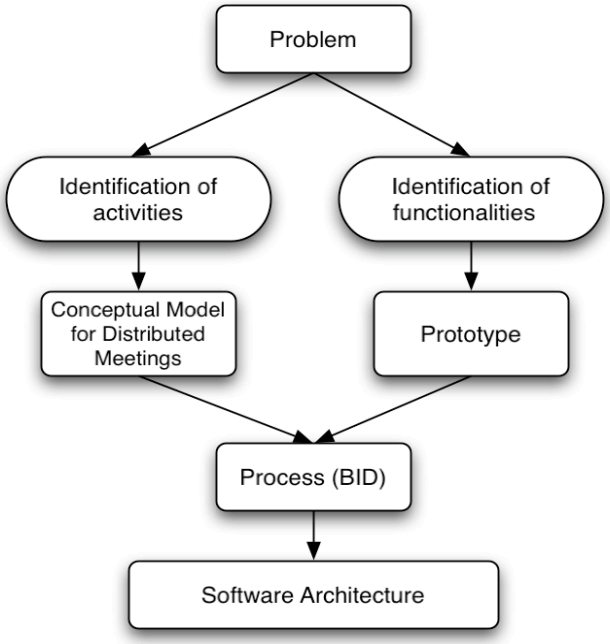


Figure 3. 5: The design process of the framework, used in the thesis

The different components of this process are described as follows:

Conceptual Model . Here we determine what kind of activities people conduct when designing software in distributed settings. We first identify them, then describe them via models. Therefore, this part focuses on identifying an abstract model describing the activities of a distributing brainstorming meetings. The models serve as a basis to build a prototype application GroupUML.

Prototyping. The prototype GroupUML plays several roles. First, through its deployment in real experiences of distributing meetings, we can observe how participants behave. Logging their activities helped us to

identify common behavior of the participants spread over different locations. Second, the prototype is used to design the prototype itself so the design produced served both identifying the requirements for features needed by the participants, and improving the design of the tool. The newly available designed architecture is then used to improve the prototype.

Process. The framework supports the activities as described by the model-activities of software brainstorming and design (described in chapter 4), which in turn contributed to improve the model itself, and finally to identify an approach for developing similar systems. The approach has evolved to a process of guidelines of how to design software systems for distributed activities. We call this process the Bootstrapping Incremental Design process (BID).

Software Architecture. As a final output of the design process, we provide a software architecture as well as a run-time infrastructure. The software architecture was built taking into consideration the activity model for software brainstorming and design meeting and the issues of distributed collaborative software brainstorming meeting activities (described in the next chapter 4).

3.4. Research Hypothesis

Our hypothesis is that distributed brainstorming and software design activity can be conducted according to the vertical process model described above.

CHAPTER 4

A Conceptual Model For Distributed Modeling Meetings

In the following we describe a distributed brainstorming scenario involving several software developers, Patrick, Jane, Allen, and John. They are located in different sites and have scheduled a set of meetings to brainstorm ideas, to sketch out and identify requirements for a software system, and to design the software architecture using UML. They can not meet in person.

In each site, the developers have a networked desktop computer and electronic white boards. GroupUML is installed on each used computer. Communication happens via GroupUML. Patrick is located in one site in France, Jane in another site in England, Allen and John are located together in another site in Germany.

Patrick and Jane are using computer desktops while Allen and John are using an electronic White board to talk to each other. GroupUML enables them to share a workspace based on the white board metaphor.

Everyone is connected and ready to start. Only Jane has excused herself and informed the meeting participants that she will join them later.

Patrick has already posted questions about the initial goal of the meeting.

Allen and John are trying to answer the questions by sketching some ideas. Patrick is pointing to a UML class created by John and has attached a question to. The question is automatically posted into the shared chat area where all participant can see who asked which question. Allen creates a different description of the problem using UML objects. Now the participants have different alternative views of the solutions to the initial questions.

Patrick, still not convinced of the current available alternatives, scrolls down the shared view and creates his own model not to overlap with the already created models. Allen and John realizes that Patrick is not looking at the same work area. They can do this through the radar view showing the whole workspace, the hidden and the visible views. Allen scrolls down to check the hidden view, and they find out what Patrick has built. Patrick tries to convince John and Allen of his alternative solution, attaches an explanation to his model.

Jane now joins the meetings, she connects to the system and retrieves all data created by the participants. Her picture is added to GroupUML so everyone is aware of her presence.

Jane examines what the others have so far accomplished. To understand why such models were created, she uses GroupUML replay feature to watch the animation session and the history of the meeting in speed mode but without replaying the audio. She watches how the different actions were conducted and by whom as if she has attended the meeting from the start. The reason for each creation and its author is shown with the models. This makes it possible for Jane to follow and grasp the contents of the shared workspace where the group memory is displayed and every step is described in the history window.

At the end of the meeting, several ideas in the form of sketches (a brief description of unfinished informal drawings), scribbling, UML models are created. Everything is saved for a subsequent meeting. The following meeting is planned two weeks later.

In the following meeting Patrick has already cleaned up the shared workspace from the scribbles that are not needed any more. After all participants join

the meeting, they start discussing what has been done in the previous meeting. John forgot to attach the rational information to some sketched ideas he created and now he has to try to remember them to explain why he decided for that idea. Otherwise his idea can not be considered when consolidating ideas.

Now it is time to rework the models created in the previous meeting session. Allen moves to building complete UML models, transforms all his scribbling into objects and classes. While refining his ideas, Patrick locks some of his models to prevent the other from changing them the time he is transforming the models.

At the end of the meeting, the created models are well structured, documented and sorted into groups of alternatives. The subsequent meeting is planned in three weeks.

Third Meeting. *Conflict identification and resolution*.

Allen is to leaving for one week to another meeting that take place in a neighbor country. Nevertheless he can still join the third d meeting with the rest of the developers from his place.

During the second meeting, Patrick sketched some ideas that present conflicts to the ones of Jane. Jane was at that time building a detailed model and is not aware of the problems caused by contradicting ideas.

Allen retrieves the models from the repository. When the meeting starts, Patrick mentions that during his offline reviewing of the shared workspace, he notices that Jane's models are presenting some conflicts with his. So they decide to solve the issues encountered by using the table of 'option against criteria' of the GroupUML. It is used to evaluate which alternative to consider according to the criteria and argumentation of each participant.

New ideas are found in the meanwhile and the corresponding models are created. So participants are just brainstorming as in the initial meeting. Then, the new models are incorporated into the already developed models. The newly added sketches are transformed to UML models. Conflicts happens again when the participants have to discard the ones that caused problems and transformed

the ones that are necessary for the final models.

Eventually, however, participants succeed to agree on to a final set of models that meet the requirements they identified during the initial meeting.

The fourth meeting is intended to wrap up the previous meetings. The aim is also to review of decisions, examine open issues, and eventually close them. Unfortunately, a last-minute change in the requirements pushes the participants to review some models of the previous meetings. Now they have to jump back to the creation of models and ideas. Some new models are created as they have enough understanding of the issues from the different meetings. After jumping from one activity to the other, participants agree upon the initial architecture of the system they designed. The meeting ends up with restructuring and documenting the rationale captured during the meetings, to serve as a basis for subsequent discussions, and the final architecture of the system developed.

The above scenario describes roughly how a distributed brainstorming and software design is conducted. Of course, we can't assume that such a scenario can help to identify all the needed requirements. The above elaborated scenario activities are reflected in the brainstorming meeting model described later in this chapter. In the following we describe the preliminary functional and non functional requirements to support such a scenario.

4.1. Requirements

We have to consider two main aspects in identifying the requirements of a framework to support such a scenario. First the framework must provide the necessary collaborative functionalities for group software design. Second, since we design for reuse, the framework must be flexible to be customized to support applications that are similar to those supporting distributed synchronous meetings.

In the following we present detailed description of the functional requirements of both aspects.

Collaborative Work Aspects

The main area of functionality of the framework is to provide support for *group activity*, *Group memory, Group awareness, Floor control, Multi-user interface, Communication, Sharing artifact,* and *Rationale Knowledge management.*

Multi-user Support. The framework must enable multiple users from physically dispersed locations to connect and use the system simultaneously.

Group activity. The main activities to be supported are object-oriented software brainstorming, analysis and design. The system must provide the necessary tools to support the management of the UML artifacts such as creation, deletion, copy, paste, undoing, and redoing modeling actions of multi-users. The system must enable the support of creating mixed formal artifacts (such as UML) with informal artifacts (such as sketches and scribbling). In addition, it must enable the export of the UML models (to a format such as XMI) to be used in other UML editors, and export the models containing informal artifacts as well. The creation of multiple versions of alternative models must be supported.

Group memory. The framework must enable multiple-users to share and enter their questions, notes, thoughts, and comments into the shared workspace. This information can be attached to an artifact in the workspace (such as a UML class) and can be stored as a whole. Each recorded information and its related models become the group memory and knowledge accumulated for the users.

Group awareness. The framework must provide capabilities to identify individual users and track their status; their actions, and their history. It should support several levels of awareness information such as informal awareness and workspace awareness. Informal awareness provides information about local and remote meeting participants (such as user's picture and name, status active or inactive) to guide their work.Workspace aware-

ness provide up-to-the-minute knowledge about other person's interactions with the shared workspace and with users (such as current user's focus, activity, and task) to help them coordinate their work.

Floor control. To prevent conflicting operations that may happen when users access shared data simultaneously, floor control is a mechanism that is used to define which user has access to which resource. It coordinates the relationship between different users performing the same activity. Floor control is crucial in synchronous groupware used for joint drawing where potential problems associated with interleaving input events from multiple users may happen. The framework must provide support mechanisms to manage floor control policies such as locking mechanism and its level of granularity. The level of granularity can be defined for an artifact, a group of artifacts, a view, or the whole workspace.

Multi-User Interface. The framework must provide a user interface that supports and manages multi-user tasking and maintain What-You-See-Is-What-I-See (WYSIWIS) mode or a relaxed one among distributed users, that is, synchronization of the users's views of the shared data model. Moreover, providing a flexible management mode to access and act on the shared data, for example, allowing deletion of artifacts created by another user, and to undo and redo all actions.

Communication. In distributed meetings, communication takes place explicitly/directly, by means of explicit such as using phone or chat tools, or by implicit means of manipulating shared artifacts. The framework must provide communication means such as chat or multi-cast audio.

Sharing artifacts. Facilitating sharing of artifacts among the group members is necessary to enable collaboration over models.

Rationale knowledge management. The framework must provide tools to record, structure, and reuse rationale knowledge.To achieve these benefits within distributed modeling meetings, significant challenges must be

met:
• designers must be able to express their design reasoning in a natural way, as it is generated during design meetings
• The representation must be formal enough to be processed for classification, browsing, and enabling search methods.
• The representation must support concurrent and distributed authoring.

Pluggability of Components

The framework has to cope with several non-meeting related issues and requirements such as *platform-independence, extensibility, reusability,* and *location transparency.*

In the following we describe these requirements.

Platform independence. The framework must support its deployment on different platforms. This is because it is rarely that different users located in different organizations share the same operating system or hardware.

Extensibility. The framework must provide support to enhance its functionality and extend its capabilities by providing explicit hook methods and abstract classes that allow applications to extend its interfaces described through contracts.

Reusability. The framework must provide interfaces to enhance reusability by defining generic or semi-generic object-oriented components that can be reapplied to create new or derived applications.

Interoperability. The framework must enable remote components of a system to interact transparently on different networked platforms.

Distributed and multi-user systems are likely to suffer from various expected and unexpected types of failures. This is due to network, hardware failure, or a non-robust component added to the system. The framework should be able to handle the following non functional requirements:

Fault Tolerance. The framework should enable the recovery from session failure. A centralized-like architecture style should enable different clients to reset their session and retrieve artifacts from the repository server.

Scalability. Although the framework is intended to support small group meetings, it should support and manage any number of users that connect to the system. The resources such as memory and network bandwidth are the physical limit of the system.

Late joiner Support. Participants must be able to join and an ongoing meeting. and grasp the content created by their peer using a replay functionality. It should allow a user who didn't attend the meeting or simply joined it late, to replay the session to see what have been done so far, who did what and who was present. It should show all the activities the models went through till the current status of the brainstorming session.

Variable WYSIWIS. The framework should make possible to change the collaboration mode from tight WYSIWIS to relaxed WYSIWIS and on the fly, that is, at run-time. This is useful for users who do not want to see changes made by their peers in real-time.

Security. The framework should provide a mechanism to secure connection between different clients if the meeting is confidential.

4.2. Brainstorming and Software Design Activities Model

During meetings, developers explore, discuss, argue, negotiate, and reach decisions via compromise and consensus. Knowledge is created, conflicts are identified and resolved, and social networks are formed. A UML meta model of the software design activities is presented in Figure 4.1.

4.2.1 Software Design Exploration

During the phases of brainstorming, software design, and software construction, developers and other stakeholders cooperate on the investigation and

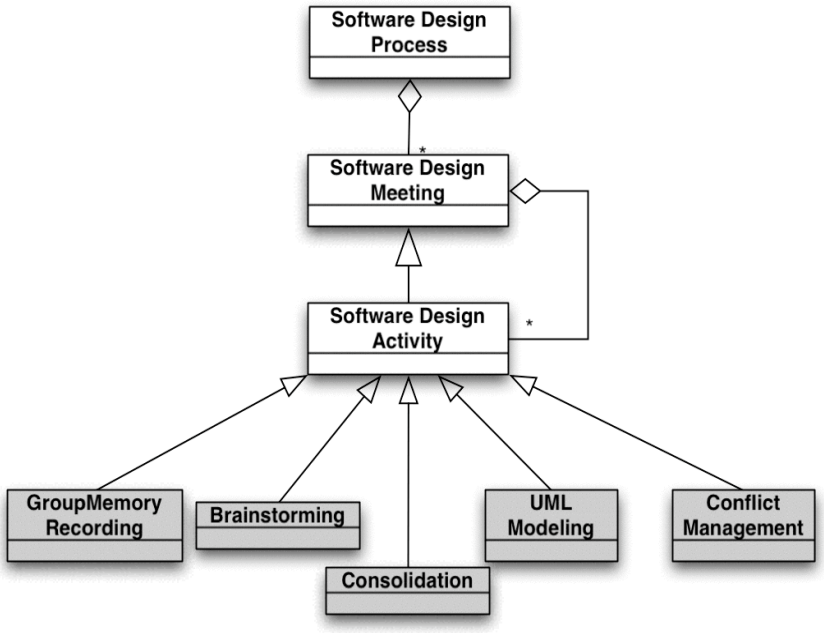


Figure 4. 1: Brainstorming and Design activities (UML Meta-Model)

identification of the requirements and functionalities of the underlined system. Because of the lack of clear requirements or ill-defined requirements, design activities often show deficiencies in the knowledge about the complete understanding of the system. Software design activities become directed toward finding, understanding, and refining the requirements of software systems.

The design activities involving systems whose requirements are well defined, are not exploratory design activities and are more about constructing a specific solution or architecture of the system. Exploratory software design [15] [143] is used whenever a complete understanding of the situation is not available. Whenever designers don't know exactly what to do next, they will engage in an exploration phase. For example, Wirfs-Brock, et al. [143], use the term ‘exploratory design’ to describe the initial discovery of objects and their respon-

sibilities during object-oriented design.

The exploratory design includes phases that are characterized by exploration of the relationship between the application domain and the solution domain, creation of scenarios of use, discovery of implementation constraints, and the generation of design alternatives.

Exploratory design allows the introduction of possible solutions before the problem is fully defined [33] by clarifying the problem and identifying promising approaches.

Developing groupware to support exploratory design constitutes an important design problem in its own right. To maintain the flexibility of exploratory design, such groupware must feel as natural as a white board as stated by Karat et. all in [75], Rosson et.al in [124], and Krueger in [78].

While several tools suitable for use in exploratory design have been developed in other context, Winograd points out that there is a need for integrated environments that support software design [142]. Tools that could be integrated in such an environment include: knowledge tools that offer alternative designs where appropriate, responsive prototyping media, design language support tools, tools to help determine user conceptual models, and tools to facilitate communication between collocated and remote designers. Such integrating design environments are not available, however some groupware such as ConversionBuilder or Grove provide limited support to facilitate software design by enabling communication among designers and users as described in [134, 78, 1].

A. Newell & H. Simon in [1] states that humans do not start think in a linear way, but rather think and act in a non linear way. Figure 4.2 depicts the initial conceptual model of software design activities.To validate it we conducted several experiments and videotaped them to check if really people start the way we describe it or just do different things.

In the following we provide a detailed description of each of these activities.

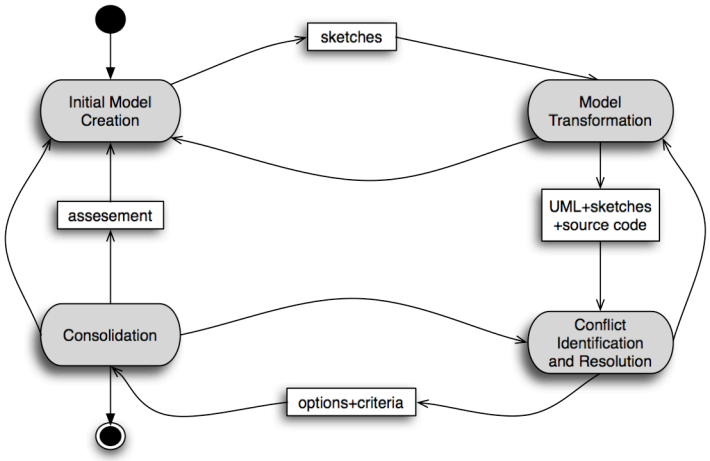


Figure 4. 2: Initial brainstorming and software design activities.

4.2.2 Initial Model Creation Activity

During this activity, participants propose and explore a broad range of solutions using sketches such as white board drawings to create an initial model. In distributed meetings, creating a shared understanding of the initial design model happens by overcoming the barriers of the physical distribution through fostering communication and making possible the discussion of ideas among developers and the generation of alternatives. In a distributed settings, support for sharing such drawings and being able to point at various details is needed.

4.2.3 Model Transformation Activity

Once a sufficient number of ideas have been explored, participants detail a small number of promising ones and eventually refine them through several transformations. Transforming available source code through reverse engineering to produce a model is part of this activity. The model is also a composition of formal artifacts, e.g. UML diagrams.

The transformation of models and the discussion surrounding the diagrams is a communication-intensive activity, everyone attempts to talk and point to dia-

grams at the same time. In a distributed setting, floor control and the ability to identify remote counter parts is needed for constructive discussions. Moreover, during the modeling activity, team members discuss several options and suggest different design views for each option.

4.2.4 Conflict identification and Resolution Activity

During design and model refinement, participants raise several issues and propose options with different argumentations.These issues need to be resolved after the evaluation of the pro and cons with respect to criteria. Due to different opinions and misunderstandings about the relative importance of criteria, conflicts often occur and need to be addressed.Therefore, structuring issues and the different options and making criteria explicit enables team members to quickly identify the source of the conflict and focus on new options to address them and agree on an objective assessment of the different options under consideration. Hence techniques to capture and maintain rationale can be used to support this negotiation.

4.2.5 Consolidation Activity

During this activity, participants review and discuss their decisions, examine open issues, and if possible close them. This activity ends with restructuring and documenting the rationale captured during the meeting, to serve as a basis for subsequent discussions.

Figure 4.3 shows a refined model of brainstorming activity as super state of all other activities. We adopted this conceptual model in designing SCOOP support to distributed synchronous software brainstorming meetings.

4.3. SCOOP Object Models

The object model of SCOOP consist of objects organized according to the Model-View-Controller (MVC) model to ensure separation of concerns. The Model objects encapsulate the application data. The *Views* objects are the repre-

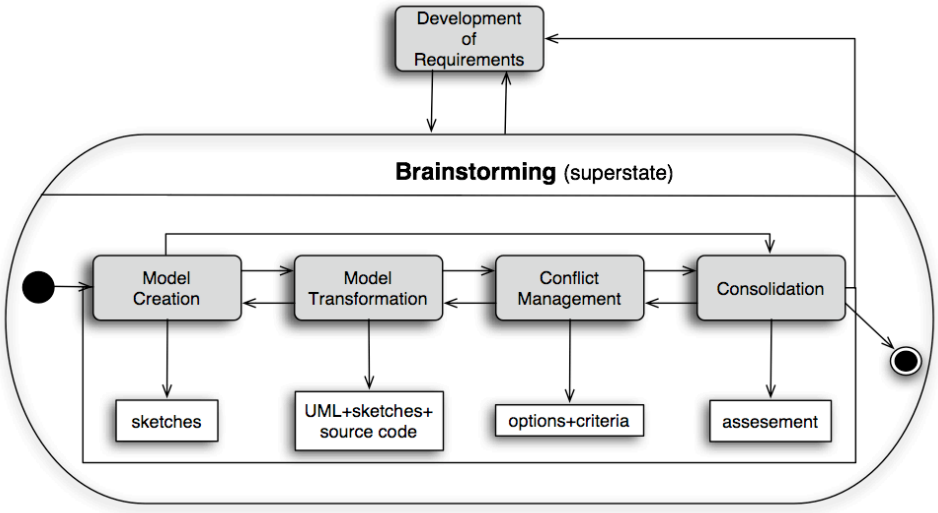


Figure 4. 3: A conceptual model for brainstorming and software design activities (UML Statechart diagram)

sentation of the Model objects, which update themselves when the model changes. The Controller objects are encapsulated within *FloorControl, GroupMemory, GroupAwareness, Rationale, Location,* and *Activity* objects. They provide various interaction methods between the user and the applications built on top of SCOOP.

Views

The SCOOP Workspace (see Figure 4.4) consists of an aggregation of WorkspaceView and WorkspaceModel objects. WorkspaceView is a mediator object that makes communication among MeetingView objects possible (see Figure 4.5). The WorkspaceModel and the WorkspaceView objects communicate according to the observer pattern.

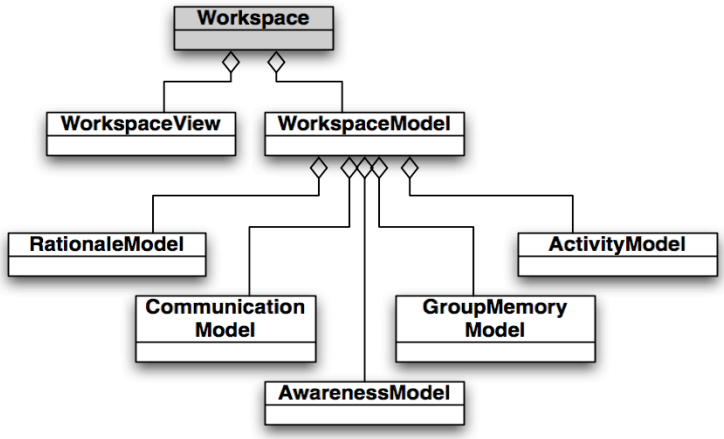


Figure 4. 4: SCOOP Workspace (UML class diagram): an aggregation of the Workspace-View object and the WorksapceModel object.

The **ActivityView** is a view containing the visual objects resulting from the activity of software design, in particular the brainstorming and software modeling activities using UML. Users build high-level software architecture, which is composed of use cases, class diagrams, interfaces, objects and so on.

An important type of ActivityView is the **ScribblingView**. It is overlaid on the UML pane and enables participants to scribble free handwritten annotations to the UML diagrams, add signs, circles, and so on, any mean that enables a communication in a team. The scribbling view enables participants to draw attention on certain details of the model and can be erased independently of the model.

RationaleView: This view consists of several related sub-views, which are associated with a rationale model. These views can be overlaid on the UML models of the WorkspaceView. An issue model is used as a way to organize meetings and diagrams and to manage their rationale. A pane view depicts the list of questions that are currently under consideration and, for each question, a

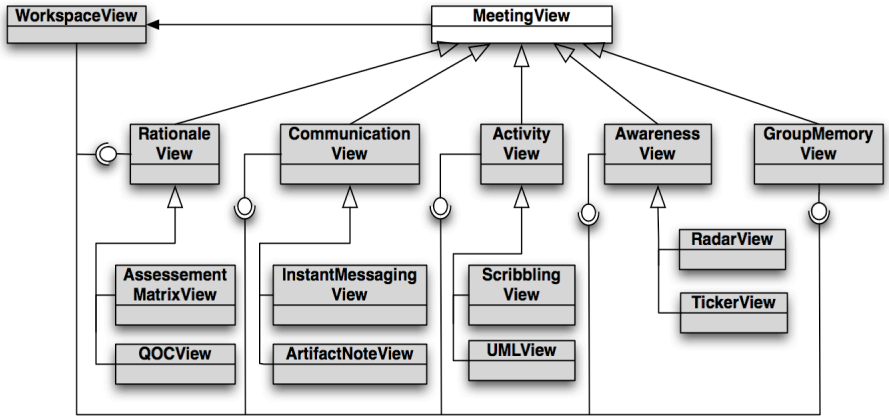


Figure 4. 5: The views object model of SCOOP: several sub-views are communicating through a mediator *WorkspaceView* object

list of options. Each option can have one or more diagrams associated with it. This enables participants to switch back and forth between different options and to pursue them concurrently. During the conflict resolution and consolidation phases of the meeting, participants use the workspace view for a collaborative argumentation to draw diagrams similar to models such as rIBIS-like [121] representation of the questions options and criteria which is translated semi-automatically to an assessment matrix to record the criteria causing the conflict and the current assessment of each option. The matrix represents another rationale sub-view that describe in a structured way the Questions that are raised while a brainstorming session, the different Options that could be envisaged and the Criteria that influence a decision. Through the assessments matrix, GroupUML supports conflict resolution activity that regroups the different options versus criteria. Criteria are used to selectively identify the acceptance or differentiation of an option. Positive assessment indicates an option satisfies a criterion. A negative assessment indicates an option hurts a criterion.

To each artifact being modeled, a related rationale view is created to track its history and the knowledge related to its existence and its relation to other arti-

facts. We define knowledge as an aggregation of the theoretical or practical understanding of a subject, the underlined facts and context (see Figure 4.6).

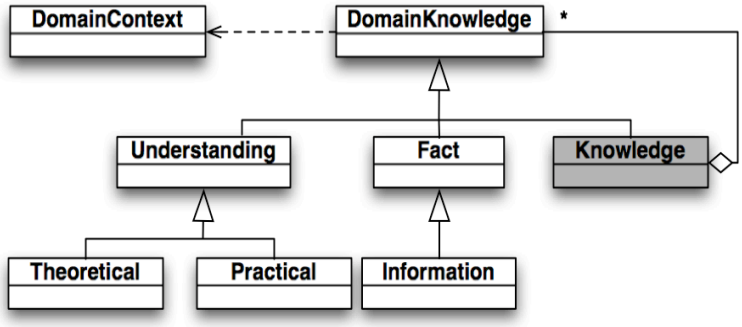


Figure 4. 6: Abstract representation of Knowledge related to a domain context

In SCOOP an artifact is a successive composition of Text objects, Drawing objects, UML element objects, and sub-artifacts as shown in Figure 4.7.

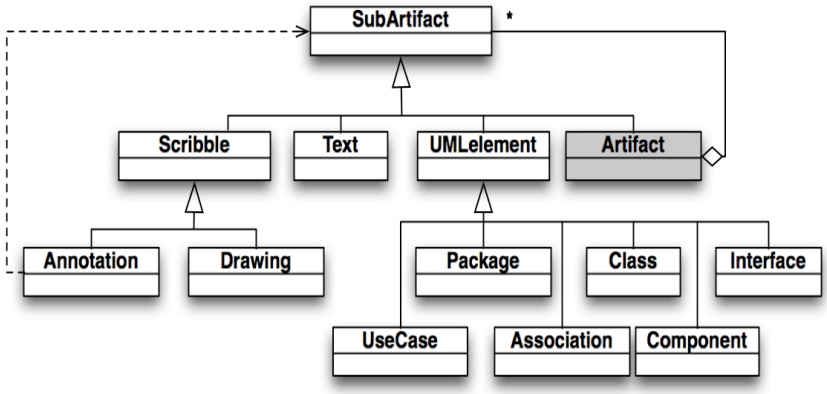


Figure 4. 7: Artifact structure in SCOOP (composite pattern)

AwarenessView. This view exhibit several aspects of group awareness such as location and activity. To every awareness information level (user, work-

space, view, task, artifact) there is a related awareness view that depicts the underlined behavior of the corresponding awareness model. For example, several users can simultaneously interact with SCOOP. Each user who joins a modeling meeting has a photo added to the list of participants. When the participant starts interacting with the tool, the photo is selected and highlighted for other sites to see. This feature provides the remote site with awareness information about who is currently modifying the diagram. This provides user awareness within a remotely working group.

Users need to know if remote users are also looking at the same artifact or model, in order to have a meaningful discussion about it. RadarView is awareness view that shows a miniature overview of the workspace to all remote users. It is tightly coupled to the ActivityView and any changes are immediately reflected. A usability study has shown radar overviews to be an effective way for people to maintain awareness of others in a spatial layout task [52]. They see changes as they occur, they know where others are working, this way, users are able to know about current focus of their peers.

FisheyesView distorts a two-dimensional space to provide the viewer with both a high-level overview of the data and fine detail around a particular location with multiple focal points to show where others are in the global context, and to magnify their area of work on all displays.

CommunicationView. Informal communication that takes place in meetings is not recorded. A CommunicationView, such as instant messaging presents several advantages such recording communication and linking it to the underlined created artifacts. The relationship between related actions like creation of artifacts are used to extract the rationale behind users activity.

A derived functionality of using instant messaging to discuss related artifacts, is that the artifacts can be used to post messages about the artifacts in the artifacts themselves. Each artifact contains a field that can be used for short messages that concern only the artifact itself. This is shown in Figure 4.8 of the communication diagram as ArtifactNote object. This way, an instant messaging view

and communication through artifacts provide participants with a simple and informal way to coordinate their actions in the case of audio quality is not sufficiently good or too many people are talking at the same time.

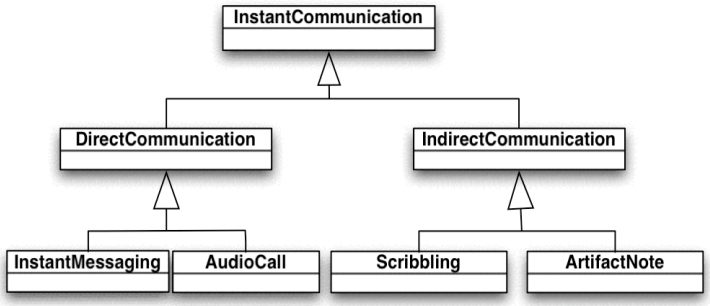


Figure 4. 8: Communication forms in SCOOP (UML class diagram')

User interaction with the workspace

In interacting with SCOOP the user can initiate two different main actions:

- Actions that change the data model: this type of action generates events that cause a change to the persistence of the WorkspaceModel (see Figure 4.4) and its visual description as well. The action includes for example a creation, deletion, modification of UML artifacts. However, this action can be undone.

- Actions that do not change the data model: this type of action does not have an impact on the data model itself. Examples of these actions are the creation of temporary artifacts that do not belong to the model such as geometrical shapes, freehand annotations and scribbles, dragging artifacts.

Figure 4.9 shows a set of use cases that describe the different types of actions that can be initiated by the user. Actions are persistent or visual. Undo and Redo actions impact the model and generate awareness events. Both actions, that is, persistent and visual, generate visual events (awareness events) that serve to attract the attention of the other users.

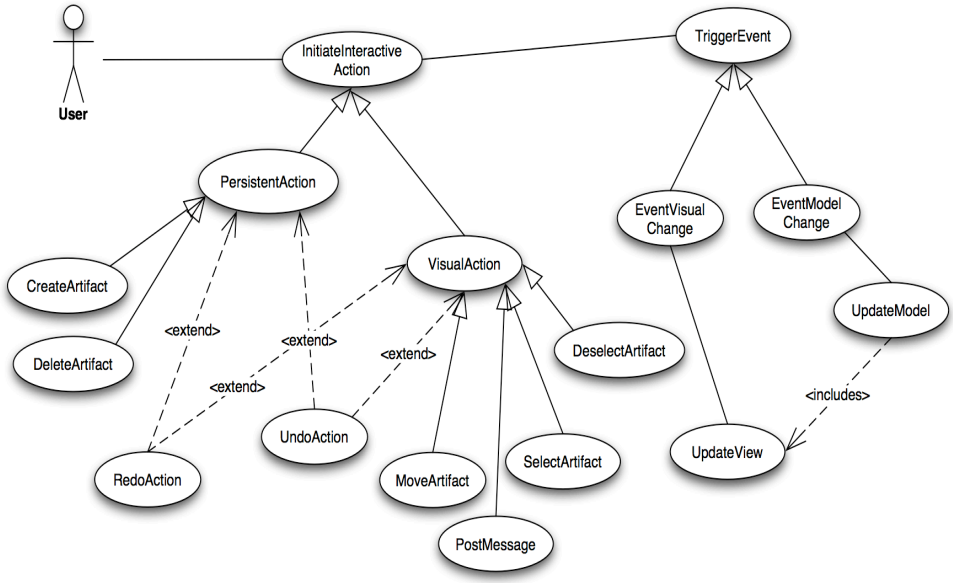


Figure 4. 9: Actions initiated by users that has different impact on models (UML Use Case diagram)

Figure 4.10 depicts a collaboration diagram showing action objects initiated by the user, which generate two different types of events: a type of event that cause an update to the model, another type of event which serves to notify the views objects, in particular the AwarenessViews, of the users action.

Floor Control

Floor control governs how participants interact in a shared computing environment whilst working simultaneously on a common task. Floor control is derived from the common social-model of turn-taking, such as the right or opportunity to speak next in debate or in a conversation. In the human-machine interaction model, floor control is abstracted into mechanisms by which access to a shared object is mediated. For example, to control access to a shared electronic white board (so that only one person can draw or write at a time) or to

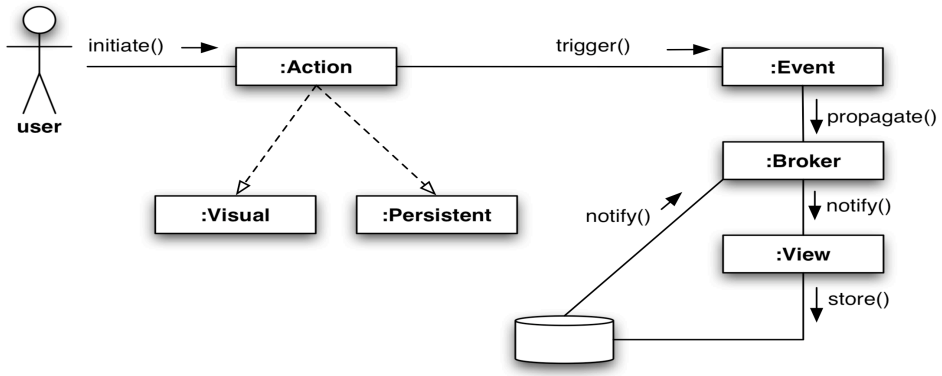


Figure 4. 10: Visual and Persistent actions: the two different actions that can be initiated by the user (UML Collaboration diagram)

determine who can speak during a chat session. In synchronous groupware systems, floor control can be applied to a particular access permission of an artifact. It refers to the right to edit an artifact (determining who "has the floor") while all users are able to view the artifact.

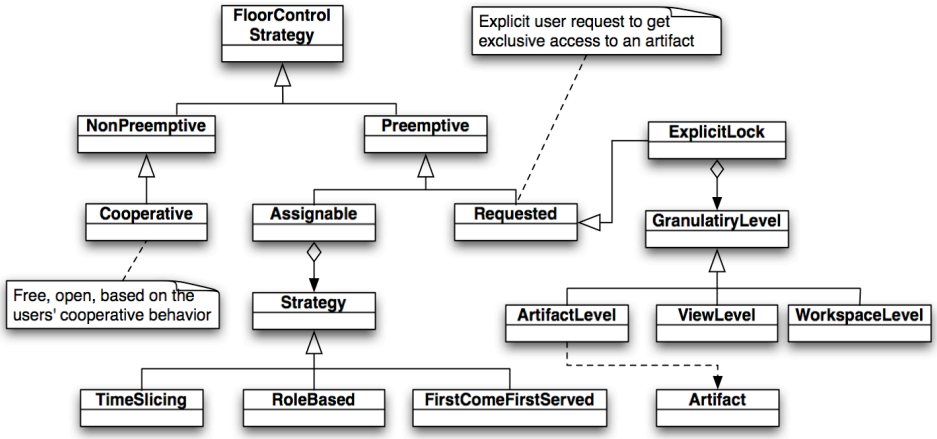


Figure 4. 11: Floor Control in synchronous activities in SCOOP (class diagram)

Participants. The floor control object interacts with the User object and the Artifact object as shown in Figure 4.11.The floor component provides lock mechanism for the User to access (edit) and control the Artifacts.

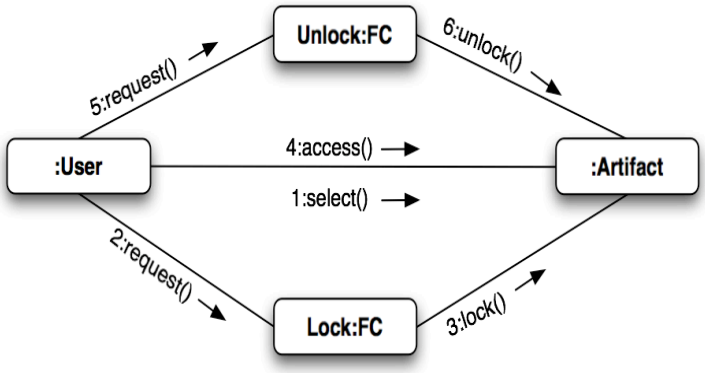


Figure 4. 12: User-Artifact interaction via FloorControl (Collaboration diagram): the user selects an artifact or a group of artifacts, requests a lock, then edit the artifact, then releases the artifact to other users.

Floor Control Policies. While we don't focus on the floor control per se, we describe briefly the different pattern behavior of its policy. Floor control policies vary from cooperative (open, free) to preemptive (assignable, requested).

Cooperative (optimistic). This is a default and simplest and easy way to handle behavior of the floor control policy. Basically any user can take the floor as soon as he or she generates input events such as selecting an artifact on the shared workspace. Users can interrupt each other at will. This is a typical behavior of people working together around a white board. As described in the chapter 6, and during our observations of the participants while brainstorming (through videotaping the meetings), we noticed that they were behaving spontaneously and implicitly taking the floor without thinking explicitly about it. Interrupting each other is quite common and accepted behavior in many cultures and groups. However, in a collaborative

groupwork environment spontaneous interruption should not abort atomic actions like creating or updating artifacts and their properties. This kind of floor control policy presents advantages in a flat hierarchy of groups or communities. However, this may lead to the inconsistency of data if there is not a mechanism that prevents it to happen.

Preemptive (pessimistic). An explicit and assignable role-based or time-slicing-based floor control policy. The moderator of the group or the administrator assign floor control level to each user according to their roles within the community or the group they belong to for example. The time-slicing-based floor control policy consists in assigning an explicit lock of the artifacts under task (see Figure 4.13 for the definition of Task in SCOOP) to each user for a given amount of time. After it expires, the next user takes the floor whether he needs it or not to access the underlying artifacts (token-ring). However, a user can release the floor if he or she wishes to.
The total time assigned to each operation should be as following: total Time = userAssignedTime + timeForAtomicTask.
userAssignedTime is the amount of time assigned to each user.
timeForAtomicTask is the amount time needed to accomplish an atomic task such as the creation or update of artifacts and their properties. It can have the value of null if the userAssignedTime expires while the user is not performing any atomic task.

This type of floor policy is interesting to deploy in highly-organized groups or communities.

Preemptive (Requested). This type of floor control is an explicit policy too. It is implemented as a part of SCOOP's user interface where an explicit lock functionality is presented to the users. Users have simply the possibility to choose the granularity of the lock and then request it explicitly. Granularity can vary from the level of artifact, a part of the artifact, a

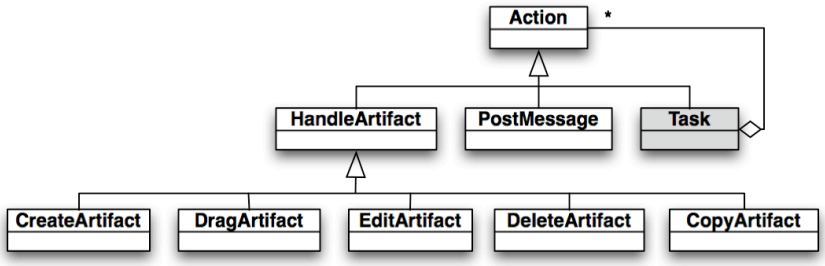


Figure 4. 13: A task definition in SCOOP

set of artifacts, a view, several views, or the whole workspace. The advantages of floor control policy are that it is explicit enough to avoid confusion of the distributed users [100]. It prevents other users to access the locked artifacts and may cause some frustrations of some users as observed during the experiments described in chapter 6. Even though, this policy guaranties consistency of the artifacts.

Location

The location object is responsible for coordinating communication between remote objects residing on different machines and possibly running on heterogeneous system, such as forwarding requests, as well as transmitting results. This object will locate the appropriate data model server, forward the request to it and transmit results and exceptions back to the client. It is used to structure calls among distributed independent cooperating objects by decoupling views objects and the data model objects that interact only through remote service invocations. The interaction among clients (views objects) and server (Model) are shown in Figure 4.14.

Group Memory

Short-term memory information reflects what happened a short while ago in a meeting.It serves to conduct several subsequent actions. Not like in real life

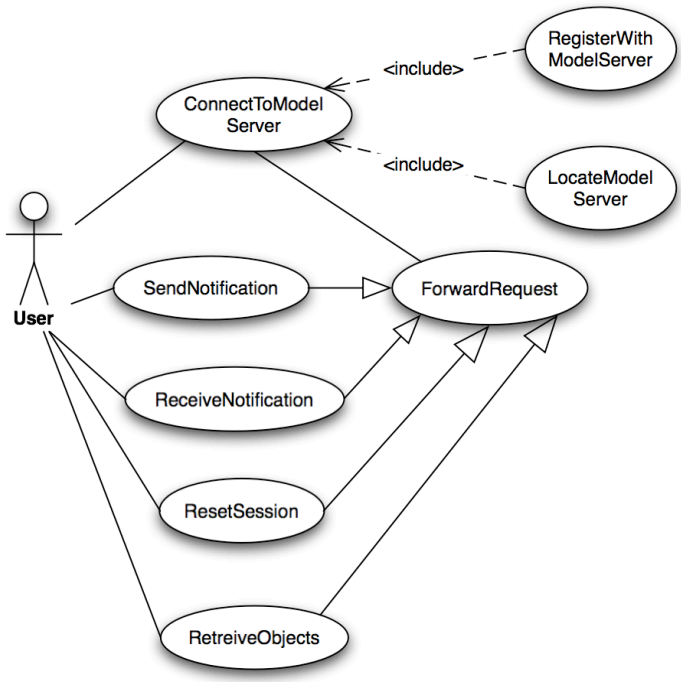


Figure 4. 14: The interaction between a client (view) and the remote server where the model resides (UML Use Case diagram)

where psychologists consider that “short-term memory erases itself” and “has a short life-time”, short-term memory in engineering groups and software system support have to be recorded for several reasons. First, short-term memory can encapsulate valuable rationale information that is crucial for further development and maintenance. Second, several short-term memory pieces can be assembled together to build long-term memory, and the history of the meeting group as well. Long and short term memories are shown in Figure 4.15. For SCOOP, group memory is synonym of knowledge that is recorded and reused each time when users face similar issues.

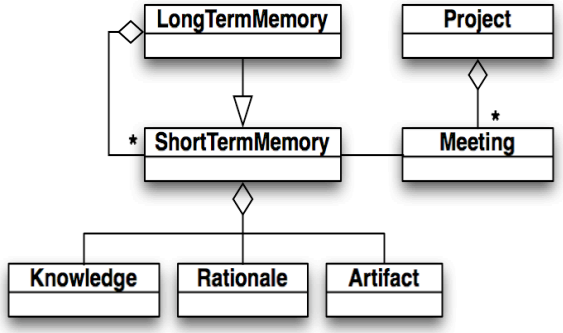


Figure 4. 15: Long and short term memory (UML class diagram)

The capture of long-term memory and short-term memory happens through several ways. Logging the users actions along the meeting, recording the rationale information, which is generated semi-automatically from the rationale objects such as questions, options, criteria, decision matrix. Figure 4.16 show the GroupMemoryCapture object and its dependency with GroupBackTrackAction and GroupMattingAction objects.

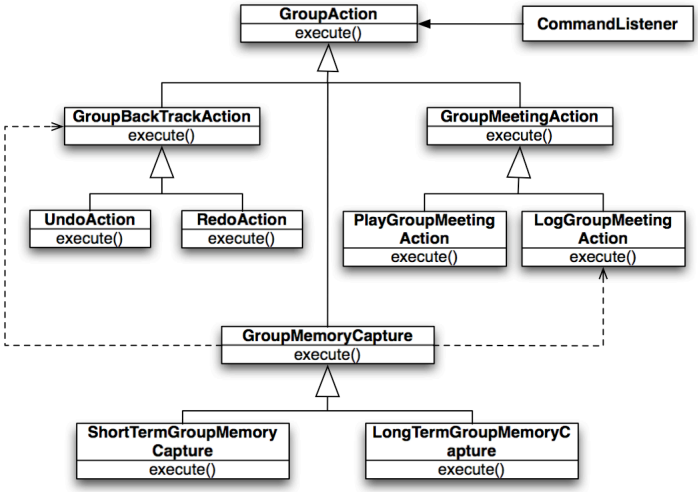


Figure 4. 16: Group Memory capture object and its dependencies (UML class diagram)

Design Activity

The software design activity in SCOOP is mainly an object-oriented brainstorming activity. It consists in finding ideas, objects, components, sub-systems, creating models, transforming models, managing conflicts, and consolidating the results. All these sub-activities are interdependent.

The activity object is transient, it used to record the state of the meeting and steps users went along the meeting. The recording serves as an input to the group memory object. Figure 4.17 shows the design activity object, the sub activities and their dependencies.

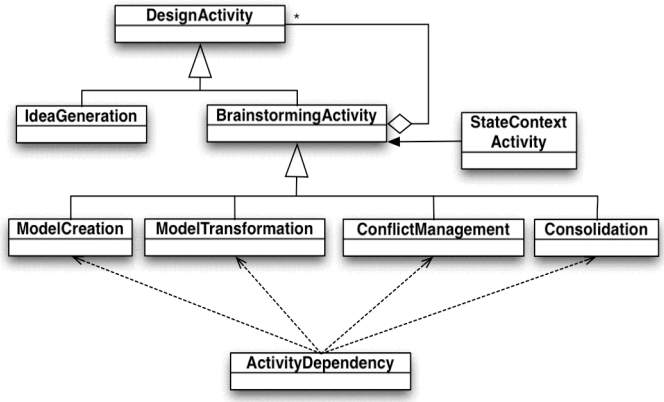


Figure 4. 17: Design activity of SCOOP (UML class diagram)

Group Awareness

Group awareness is an aggregation of various sources of knowledge and perception information of the workspace, users, their activities, and tasks.

With SCOOP, we focus on workspace awareness, which provides necessary knowledge about participants, their tasks, and their focus, which helps in coordinating their activities.

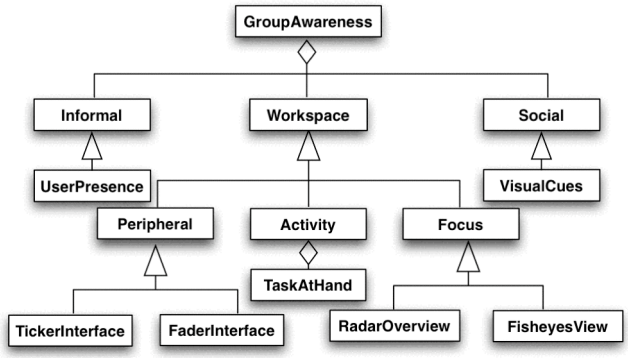


Figure 4. 18: Group awareness class diagram

Workspace awareness

Workspace awareness is participants's up to-the-minute perception of what others are doing and on which part of the workspace they are working. In distributed settings, social awareness such as visual cues are not available to all participants, especially when participants are allowed to have different viewports into a large workspace. Consequently, workspace awareness support must be provided that inform a participant about where other people are working in the shared workspace and what they are doing. RadarView and FisheyesView make showing current focus of remote users possible.

Communication

SCOOP supports real-time communication such as instant messaging, scribbling and communication through artifacts. Scribbling allows users to write and draw quickly messages that overlap other artifacts such as UML models, typed text and so on. It makes possible to connect different artifacts of the workspace that can have different semantics, such as connecting a UML class to a geometrical artifact of any form. Communication through artifacts (Artifact-Note, see Figure 4.8), is a handy way of posting messages concerning the artifact

itself. Each artifact has an editable text field that enables users to write text in, so that it is shared and viewed instantly. The message can be overwritten or scrolled down. As a consequence, artifacts are not mute any longer, and the users discussion about the artifacts is attached to the artifacts themselves.

Rationale

In investigating solutions to the issues related to Rationale management in distributed collaborative software design meetings, SCOOP uses a hybrid method for capturing the design rationale:

Process-oriented design rationale capture. Capturing rationale as it happens, through informal and /or formal artifacts where developers discuss different options, draw incomplete diagrams (high level UML or simple diagrams) and provide their argumentation,

Structure-oriented design rationale capture . Developers try to identify and resolve conflicts through available criteria and options using ICAA (Issue, Criteria, Argument, Alternative) graph synchronized with an assessment Matrix for design decision [101]. The argumentations ensure the capture of design rationale. Hence design-decisions and design rationale are linked together and evolve simultaneously.

Post-mortem design rationale capture. To avoid design process disruption, developers can delegate the management of the structured design rationale and modeling spaces to the facilitator that fill out and restructure the ICAA

space whenever needed.

Question	Criterion C1	Criterion C2
Option O1	+	-
Option O2	-	0

Table 4.1: The assessment matrix in SCOOP: The options O1 and O2 are evaluated against criteria C1 and C2. the '+' value means that option O1 meets the requirement C1, but not the requirement C2 shown with the '-' value. '0' is used for not applicable or neutral.

SCOOP uses an argumentation-based rationale management, which is an approach that represents rationale as a graph of abstract steps, also called issue model (see Figure 4.19). Structuring issues and the different options and criteria enables users to quickly identify the source of the conflict and focus on new options to address them [102]. A simple representation of the rationale is the QOC (Question, Option, Criteria) [10], which can be represented as a matrix such as described in Table 4.1.It is used for the evaluation of the different alternative solutions and conflict resolution.

The use of the instant messaging feature enables the users to choose to check a box with a "Q:" prefix whenever they need to ask a question or to post an issue; users check a "O:" prefix when posting a message as an alternative. This is useful to make possible the semi-automate filtering of knowledge during chat and populating the assessment matrix (see Figure 4.20).

Using ArtifactNote to attach informal notes to UML elements is an efficient collaboration tool. The notes could then be used as a basis for populating the more structured issue model of the meeting. On the other hand, by attaching an Option, a Criteria, and an Issue to a UML artifact, users are able to construct formal notations during modeling in the same way they connect UML artifacts.

Issues (expressed through questions), Options, and Criteria associated with UML artifacts are inserted automatically into the assessment matrix that supports decision-making through the evaluation of each option manually.

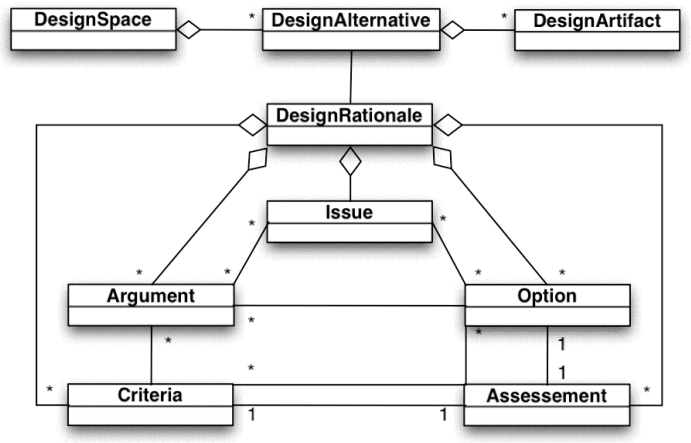


Figure 4. 19: Rationale issue model in SCOOP (UML class diagram)

Integrating rationale capture at a reasonable cost without disrupting the meeting and applying this to distributed meetings has many advantages such as:

- Developers need to identify the source of contributions, in terms of authors and roles, as they usually do not know each other
- Collaboration through a tool (as opposed to face to face), provides users with more information, such as the recordings or the logs of the meeting, to work with that can be reused for subsequent meetings.
- The cost of rationale capture or post processing might become acceptable comparing with the cost of mistakes resulting in not considering rationale.
- Distributed development meetings remove informal communication, which may lead to misunderstandings of the developers' respective goals and intentions, especially if the developers have never met before. Rationale helps make this missing information explicit.

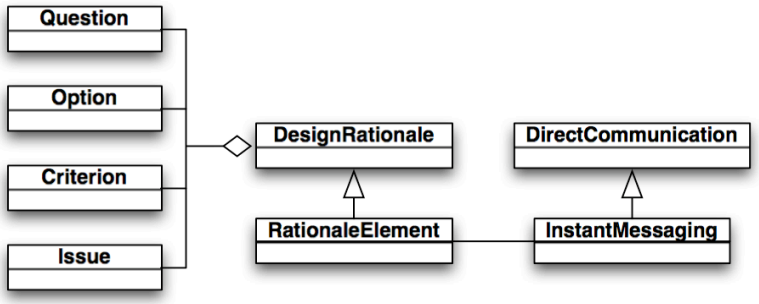


Figure 4. 20: Associating rationale elements to instant messaging and vice-versa. The association makes possible the navigation to rationale elements from instant messages and the navigation from the instant messages to the rationale on the diagrams.

Objects collaboration

Figure 4.21 shows a summary of all SCOOP objects and their collaboration. The summary shows also a full-cycle of collaboration among all objects after a user initiates an action on the workspace.

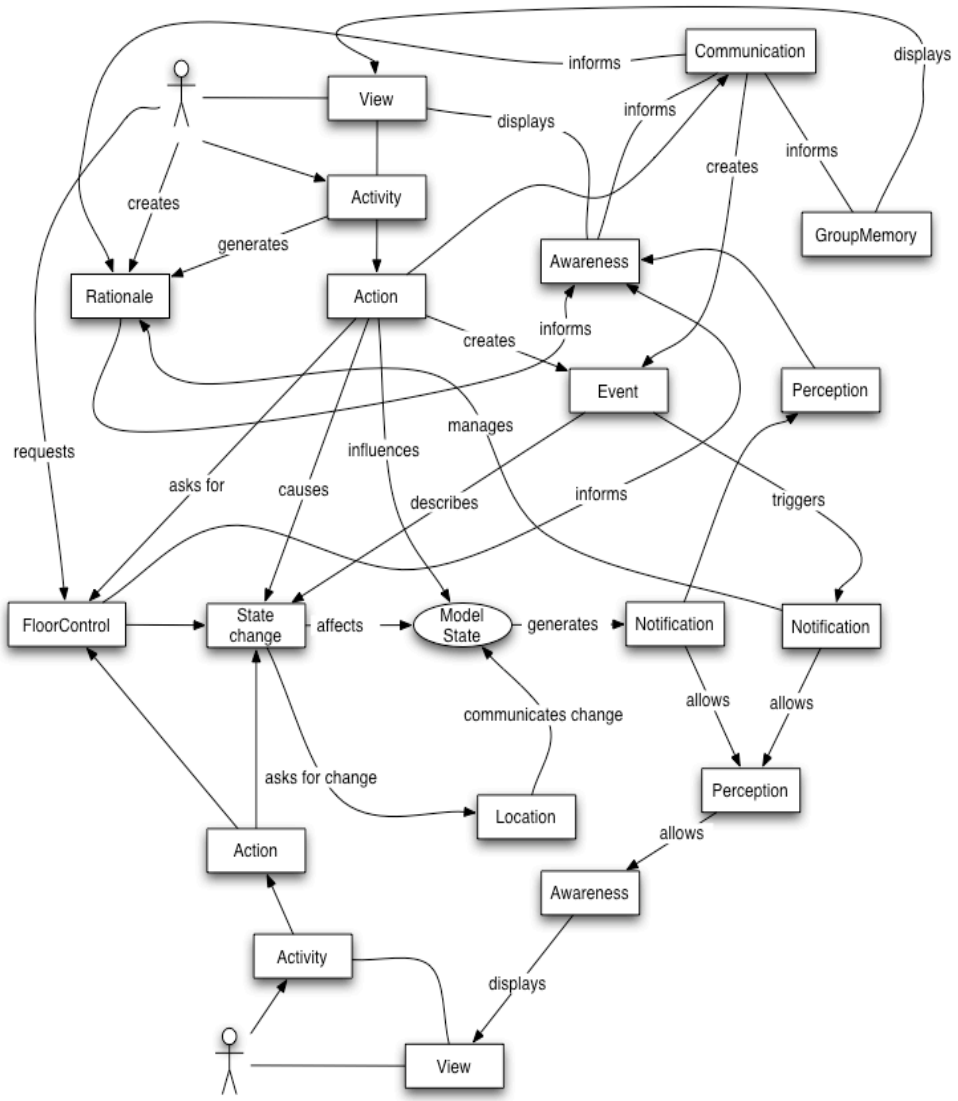


Figure 4. 21: Summary of all SCOOP object showing the full-cycle of their collaboration and interaction after initiating an action by the user (Collaboration diagram)

CHAPTER 5

The SCOOP Framework

5.1.SCOOP Framework Development

The design of any type of framework requires the consideration of many issues. Although frameworks have several benefits such as design reuse, there are also drawbacks, such as the increased cost of building a good object-oriented framework as compared to a single application.

The need to our framework has arosen from examining the domain knowledge of the early stages of distributed software development. In particular to setup distributed meetings of brainstorming.

Brainstorming necessitates concurrent input and collaborative thinking. Brainstorming meetings are difficult to manage because of the way people think and react to counterpart ideas. People's behavior and the amount of ideas generated vary within a meeting and from one meeting to another. Therefore, the quality and amount of knowledge produced during a meeting is variable. As a first consequence, artifacts produced during meetings have to be managed and dealt with for subsequent meetings.

5.1.1 Component-based Framework Design

SCOOP is designed as a collection of reusable component objects [107]. We emphasize on building reusable components because this is a requirement for the flexibility of SCOOP framework. The concrete classes provide the reusable components while the design provides the context in which they are used.We use contracts to describe the behavior of components within SCOOP framework.

The reasons of using components are manifold. Using components that are built and tested and verified reduce the risk of using code written from scratch. Components that are mature present higher quality and reliability then writing own code. Through generic interfaces, components can be adapted to cope with special requirements of systems having similar domain applications. This reduces the effort of development and makes possible to keep focus on more difficult issues then coding.

Contract-based components of SCOOP

"The main purpose of contracts is to help us build better software by organizing the communication between software elements through specifying, as precisely as possible, the mutual obligations and benefits that are involved in those communications." Betrand Meyer [11].

We concentrate on describing contracts of high-level components and their underlined design rather than classes. We see contracts as a useful formalism to express high-level specifications of components behavior, because it facilitates the management and reuse of the SCOOP's components. They are the replaceable part of SCOOP that conforms to the well defined interfaces.

Different implementations can be provided that show different behavior (e.g., different locking mechanisms within the same application) or adapting a component to fulfill a special requirement in an application having the same domain as SCOOP.

Figure 5.1 shows the components that make up SCOOP.

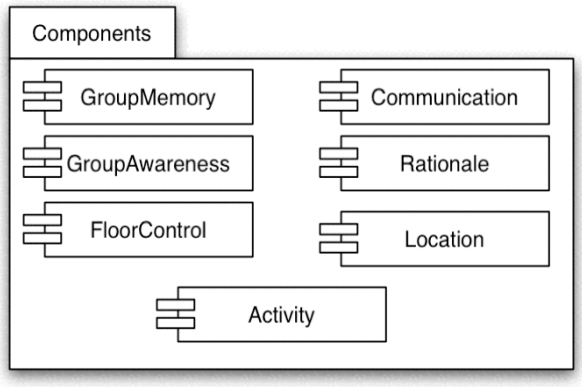


Figure 5. 1: SCOOP Components: during the requirements analysis we identified these components

A formal presentation of a components meeting model (as described in chapter 4) using the BNF notation [13]:

<Component>::=<Activity>|<Communication>|<Location>|<Rationale>|
<FloorControl>|<GroupMemory>|<GroupAwareness>
<Activity>::=<Design>|<ConceptMapping>|<MindMapping>
<Design>::=<ObjectOrientedBrainstorming>|<Modeling>
<Communication>::=<Scribble>|<InstantMessaging>|ArtifactNote
<GroupAwareness>::=<InformalAawareness>|<SocialAwareness>|
<WorkspaceAwarenesss>
<WorkspaceAwarenesss>::=<PeripheralAawareness>|
<ActivityAawareness>|<FocusAawareness>

<FloorControl>::=<Preemptive>|<NonPreemptive>
<Preemptive>::=<Requested>|<Assignable>
<Assignable>::=<TimeSlicing>|<RoleBased>|<FirstComeFirst-

Served>
<NonPreemptive>::=<Cooperative>
<Location>::=<Distributed>|<Local>
<DesingMeeting>::=<Location>{<Component>}

In the following we describe the main components of SCOOP as outlined in Figure 5.2:

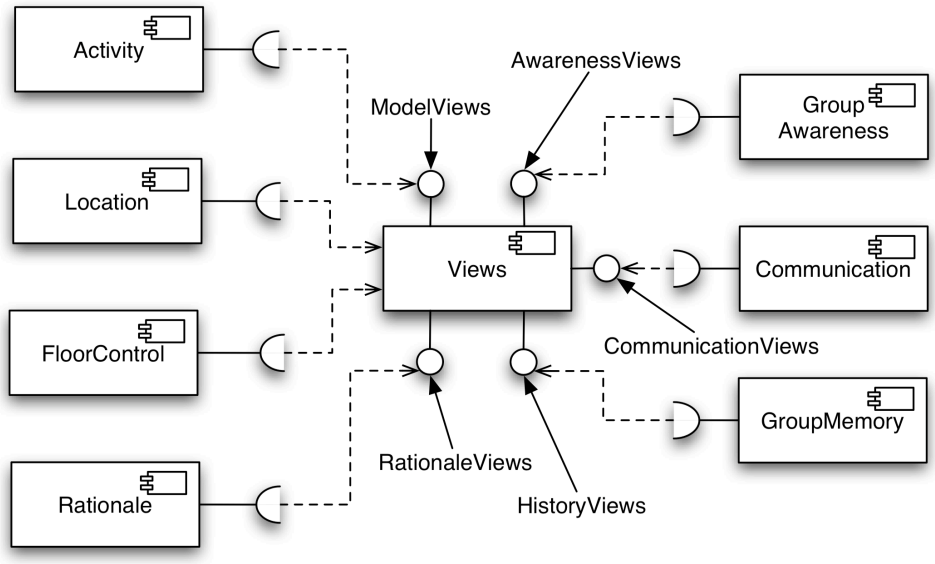


Figure 5. 2: Components of SCOOP: the views component provides interfaces to talk to each component. Each component updates its corresponding view through its interface.

Floor Control component requirement and considerations

As described in the analysis model for SCOOP (see chapter 4), this component encapsulates the behavior of floor control policies. It is intended to be easily substituted for other components that meet the same specification.

Hence, different implementations of floor control are provided with the framework and can be loaded as needed and according to the type of the activity conducted by the group. In a distributed groupwork, choosing a floor control policy at runtime presents several advantages for coordinating and conducting a smooth interactions between remote users. Free or open or cooperative form of floor control policy for example can be of advantage when brainstorming ideas, where every user can interrupt at will each user without asking for a floor or waiting till having explicitly the floor. Whilst transforming design models or restructuring the design space, the moderator can load different floor control policy such as based on locking artifacts under task, which presents advantages.

For developers, the floor control component presents interfaces enabling to implement and deploy new and different floor control policies within SCOOP framework.

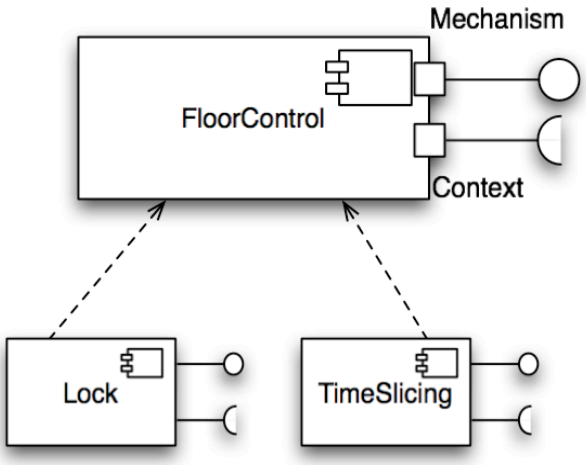


Figure 5. 3: Floor Control Component in SCOOP: the Lock and TimeSlicing components implement the generic FloorControl component.

Contract description. The contract is about the responsibilities and the needs of the floor control component.

Floor control component defines an interface for single global floor control policy. That is, it enables turn-taking within physically distributed group of participants conducting a common task. When invoked, the available policy is applied and the underlined artifact is made available only to the invoker.

Pertinent Design Pattern. The strategy pattern is used enabling floor control component to load the policy corresponding to the context provided by the application. The pattern also enables the switching from one floor control policy to the other at rune time as well.

Participants. Two main components are interacting with the floor control component: the component representing the User and the artifacts of the View workspace component. Figure 5.4 depicts the interactions among the User component, the FloorControl component, and the WorkspaceView.

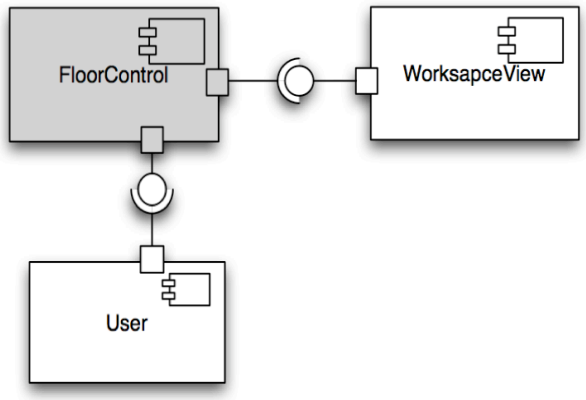


Figure 5. 4: Components interacting with the FloorControl components (UML Components diagram): The Workspace component provides the artifacts that can be accessed using the floor control strategy provided by the context of the application. The User component uses the interface provided by the FloorControl to regulate the access to artifacts of the WorkspaceView.

Invariants. In a synchronous groupwork, floor control policy provides mechanisms to manage and control access of the underlined artifacts. The floor control policy maintains the consistency of the artifact, that is, changes made by a client under the use of current floor control policy, are not lost and can not collide with other changes made by other clients. As a result only one consistent copy of the artifact is available at a time.

In SCOOP we experimented with two different floor control policies. First, an optimistic floor control strategy, which allows users to manipulate objects without locking, relying on socially accepted practices. Users take their cue from the facilitator and awareness mechanisms to synchronize themselves and facilitate the meeting. This approach allows supporting different meeting styles and avoids disrupting the flow of conversation by requiring users to perform additional actions (e.g., explicit locks). Second, the pessimistic floor control policy of locking, where users can lock an artifact, a group of artifacts, a view, or the whole WorkspaceView. Note, that the lock policy implements a concurrency control within SCOOP that resolves concurrency hazards and ensure that the shared model is the same for all users.

Location Component requirement and considerations

This component is responsible for making possible the communication between remote components residing on different machines and possibly running on heterogeneous system. It makes inter-object communication transparency possible, such as forwarding requests, as well as sending back results. It is used to structure calls among distributed independent or loosely-coupled cooperating components by decoupling ViewsComponents and the data ModelComponent, which interact only through remote service invocations.

Contract Description. The contract of the location components describes the responsibilities of the views and their needs. The location component is a broker component. Its role is to enable inter-remote-objects invocation to happen in a transparently. This behavior achieves better decoupling of clients objects

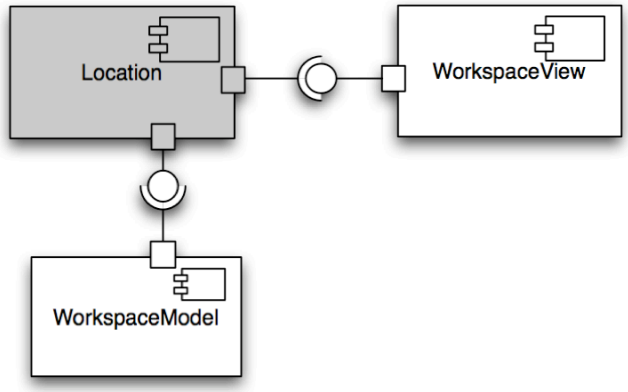


Figure 5. 5: Location Component and its interaction with the WorkspaceView and WorkspaceModel components

and servers objects. yet, the location component ensures the location transparency behavior as an important requirement to enable distributed synchronous meetings to happen as the same simplicity as in single-site meetings. the ModelComponent register itself with the location component and make its service available to client objects, that is the views components. The views components access the functionary of Model component by sending requests to the location component.

Pertinent design patterns. The Broker pattern [42] is specialized case of the mediator pattern [40]. The only difference is that it coordinates communication between distributed objects instead of single-side located objects. Objects do not have to concern themselves with the details of remote communication that is encapsulated into the broker component.

The Broker pattern is used to structure distributed software systems with decoupled components that interact by remote method calls.The requirements for the location component is to enable coordinating communication between views components, and also to encapsulate the protocol and middle ware for

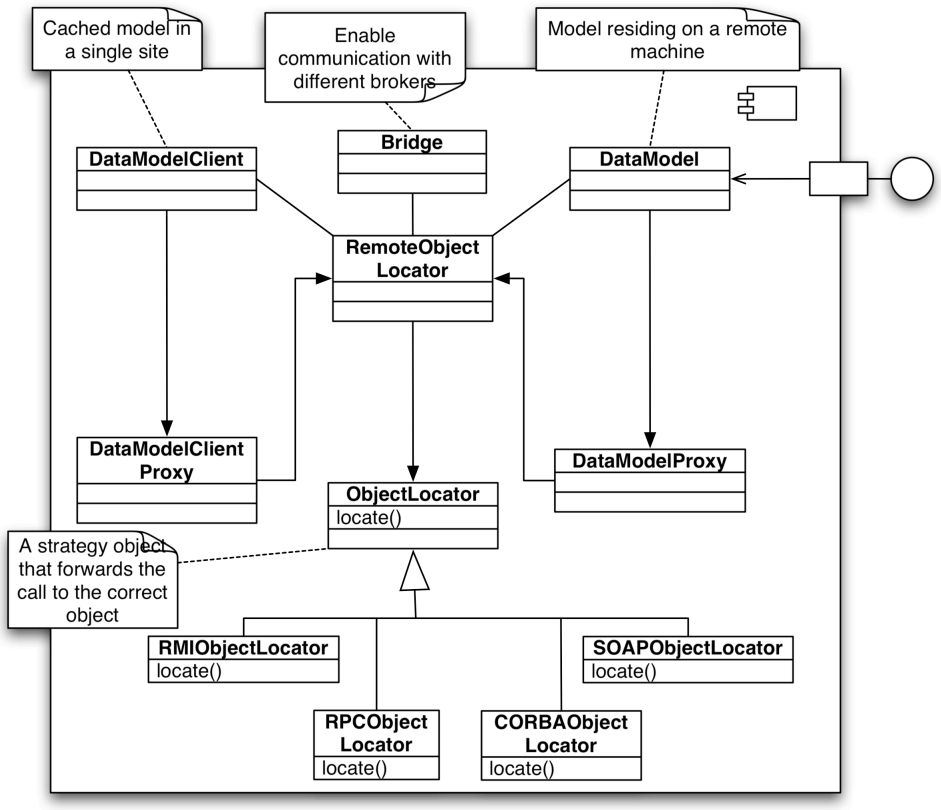


Figure 5. 6: A modified Broker pattern to which we added a strategy object that makes different implementations protocols for remote object calls possible, such as RMI, CORBA, RPC and so on.

implementing the remote call strategy through RMI protocol, a traditional RPC protocol, a CORBA protocol, or a SOAP call. As a consequence, the location component is combination of the broker pattern and the strategy pattern as shown in Figure 5.6.

Participants. The location component collaborates with the views component and the Model component.

This component encapsulates the behavior of locating remote model server, using the necessary protocol to forward the request to the server. An instance of this component is the RemoteObjectLocator component that ensures a location transparency of the remote objects and their interactions.

Invariant. Independently from the protocol and middleware used such as CORBA, RMI, SOAP, RPC, etc...., the Location component ensures to route inter-objects calls in the same way and therefore guaranties that the call reach its callee.

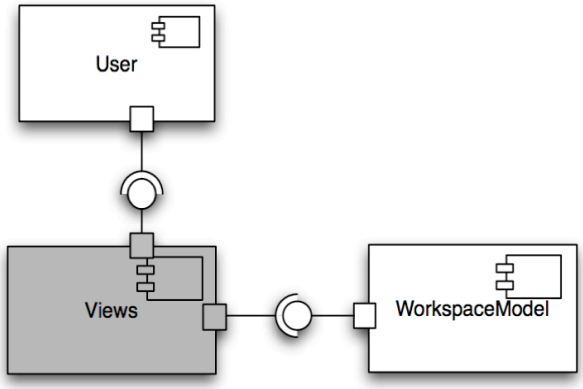


Figure 5. 7: Views Component of SCOOP interacting with the User component and the WorkspaceModel component.

Views Component requirement and considerations

The main visual view (WorkspaceView) of the framework consists of several separate single views. Views are designed as components representing the visual interactive capability of the framework. Breaking down the main view into fundamental abstract single views enables their reuse in different systems

with similar context. RationaleView (see chapter 4) for example, can be reused to support similar groupware enabling distributed user activities such as mobile maintenance where a worker can get help from a remote expert and the system can capture and show rationale information of current issues.

Independently from the used display device, all users see the same view currently being manipulated in a flexible way according to the MVC pattern (Model View Controller). Each user has control over his current view, however, all users share the same model (WorkspaceModel) and any modification of a local view is spread out to all other views. In presenting data to the users, most of current synchronous CSCW systems have adopted either a strategy of a relaxed of WYSIWIS (What You See Is What I See) or a tight one. A relaxed strategy implies that users do not see changes of the shared Model immediately but deferred or on demand. The tight strategy means that all users get the same data almost instantly. SCOOP views implement a behavior of model presentation levels that vary from a relaxed to tight at run time. Hence, different users can have different artifacts presentation strategies. As a result, users can have the same view or different description views of the same model.

All artifacts created by each user are stored in the WorkspaceModel object that resides in the model server. The model is shared across the views composing SCOOP framework.

The relationship between views components and model components are described through the observer-observable pattern. Updating Views are updated through notifications as soon as a model data change.

Participants are provided with four other view panes for collaboration that can be optionally shown or hidden.

To each view created and its artifacts, a related rationale view is created to track its history, its rationale, and its relation to other artifacts. Figure 5.9 depicts two related rationale sub-views. In the tree-like QOC representation, we notice that for example that to a given question or issue, we identify several options that meet some criteria that are attached to them. SCOOP translates the relationship

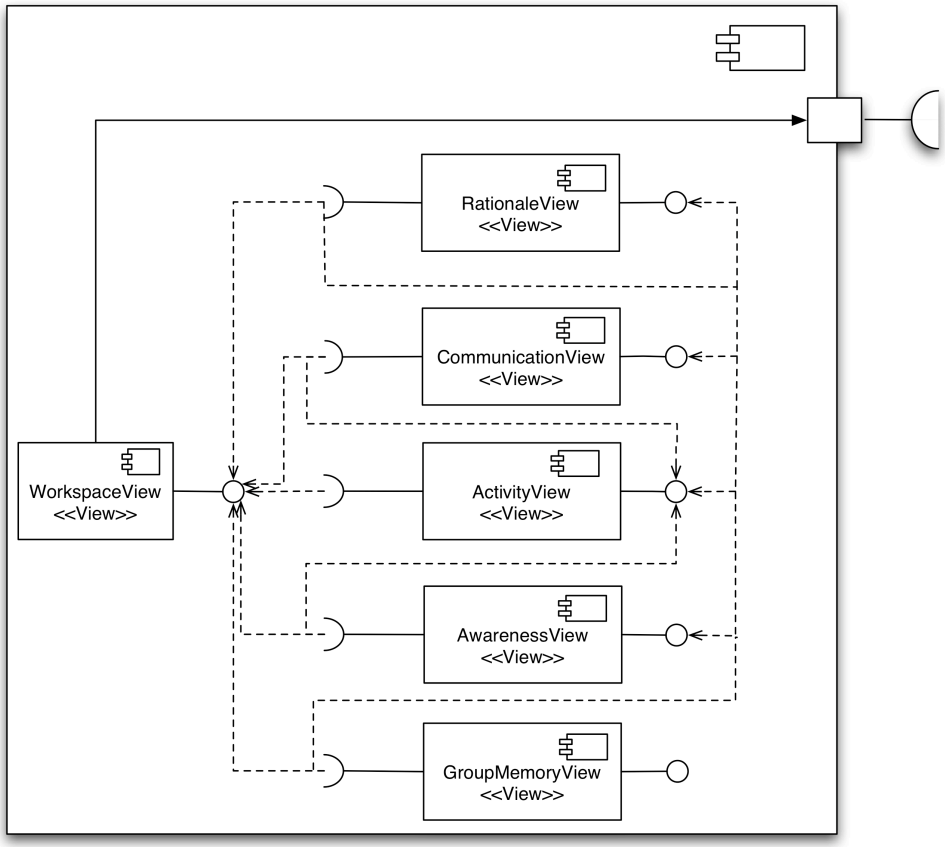


Figure 5. 8: Views components and their inter-dependencies

into the assessment matrix following the rules:

1 . Questions.

The questions and their corresponding options are entered as they are in natural language. That is SCOOP is not pre-processing users entries.

2 . Option

One or many options can be attached to each question.

3 . Criteria.

For each criterion that is attached to an option is assessed positively, that is identified as accepted, all other criteria (that are attached to other options) are assessed negatively or as non relevant or not applicable. However, according to the context, criteria attached to different options may have common background. This should be updated manually by the users.

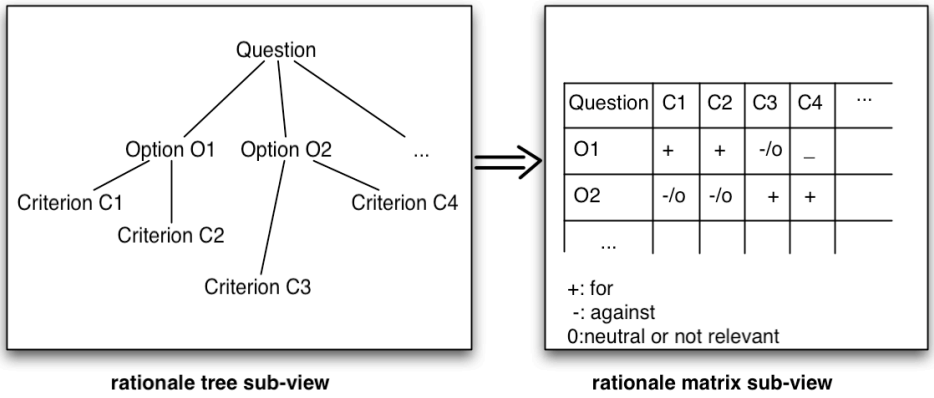


Figure 5. 9: Question Option Criterion tree to table transformation

Contract description. The contract of the view components describes the responsibilities of the views and their needs. Views update their state based on a event-driven mechanism.Views register with the model component, which plays the role of event dispatcher, and get notified whenever the model changes. The primary role of the views is to update their state and update the data model (WorkspaceModel) via the broker component after a user initiates an action.

Each view component defines interfaces and abstract classes that can be reused by developers to develop own views or to extend existent ones. Each view represents is related to the corresponding data model according to MVC pattern.

Pertinent design patterns. Several design patterns were used in modeling the views component, their inter-communication. A requirement in designing the views components is that views should be able to communicate with each, new views can be added and removed without any impact on the structure of the whole system. Moreover, the context in which the views interact can be complex and unstructured such as in distributed settings. Views describe different kinds of informations that are dependent or partially dependent on each other. For example communication views do not need to know about rationale information views, however rationale views need to know about communication and the underlined artifacts in the workspace views. Hence, their interdependencies are unstructured and difficult to understand, their communication and behavior is also complex to maintain. SCOOP uses a mediator object that encapsulates the cooperation between different views. The views do not need to know about the different views, and they do not communicate directly with the other views, instead they route their calls through the mediator

Views also need to be notified whenever the data model changes. Using the observer pattern to maintain consistency across the state of the data model component and the views components is a straightforward solution. Moreover, the data model objects reside on a different machine as the views (see Figure 5.20 deployment diagram), thereby, we extended the observer pattern behavior with remote capabilities through a broker component described in the next section.

Participants. The user component interacts directly with the views components. The user initiates actions that can impact the persistence of the model components. The model components in turn notify the views components whenever they change state.

Invariants. Adding new view components or removing them does change the data model nor influence the functioning of the system.

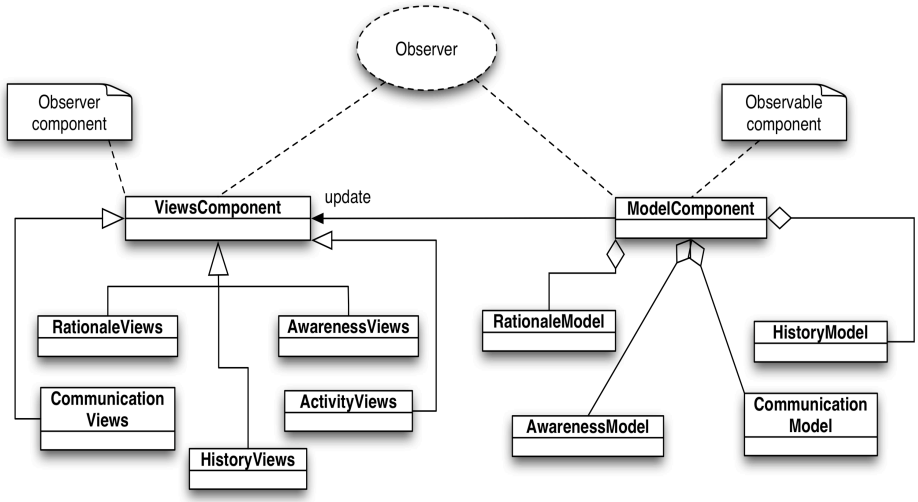


Figure 5. 10: Models and Views collaborate via the observer pattern

Figure 5.10 show the relationship between the ModelComponent, which is composed of several data models, and the ViewsComponent.

GroupMemory Component requirement and considerations

As described in the analysis model of SCOOP, during meetings, groups need memory. Short-term memory and long-term memory as well. Both are crucial to run meetings and to enable long-term collaborative support within projects where necessary information are stored to describe artifacts, their history, the rationale behind them, and the users who created them.

Group short-term memory is also the information that were said and the events happened during the meeting, which are crucial for the continuation of the meeting. When recorded, it constitutes the history of the meeting. The GroupMemory component can replay the meeting using every piece of the short-

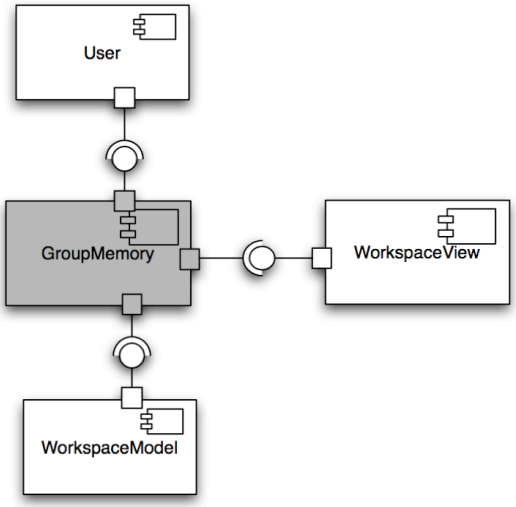


Figure 5. 11: GroupMemory Component and its interaction of the WorkspaceView, the WorkspaceModel, and the user components

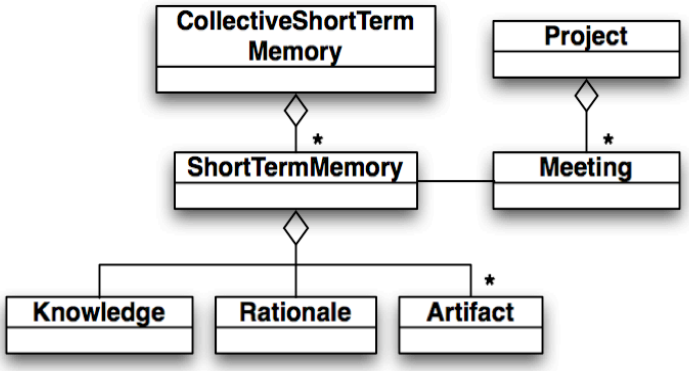


Figure 5. 12: Collective short term memory is the collection of the individual short term memory of the meeting participants

term memory information recorded during the meeting. In traditional informal meetings, meeting attendees are usually taking notes for themselves. SCOOP supports recording collective short-term memory which is the collection of indi-

vidual short-term memories that are exposed to all participants. Hence, group memory represents a wealthy knowledge produced during meetings by the users. It is crucial for long-term collaborative support within projects where necessary information are stored to describe artifacts, their history, the rationale behind them, and the users who created them. In traditional meetings, long-term group memory is generally recorded by the minute taker and will not be available until the next meeting. SCOOP makes long-term memory available and explicit to all distributed users instantly.

Contract Description. GroupMemory component role is to help facilitate the meeting by making the collective short-term memory explicit to distributed users and to record it for subsequent uses. Collective short-term memory can be used to cope with the different individual recordings. Participants write instantly all short-term memory information in front of the group, using a white or a black board. This has several benefits, first, individuals are no more overwhelmed with information, figures, and charts that are brought up, second, concentrating on own ideas only, which prevents from giving full attention to the group, is no more necessary since own ideas can be shown on the collective memory and one is open to the group's new ideas. Figure 5.12 shows the collective short term memory, short term memory abstractions and their relationships to the Meeting and rationale knowledge

GroupMemory component has the responsibility to track every single action and record it to enable users to undo or redo according to their needs. It also records the whole meeting history

Pertinent Design Patterns. GroupMemory component implements the Command pattern to enable to queue and execution of requests of undoing or redoing actions, replaying actions of creation, deletion, motion, communication etc.

Participants. The participants in making the work of this component suc-

cessfully are the WorkspaceView component, User components, and the WorkspaceModel component as shown in Figure 5.11. Through the user action, actions are performed and the result is sent to the views components to display them.

Invariants. Since GroupMemory records in background what happens during the meeting, it does not influence the running of the meeting.

Activity Component requirement and considerations

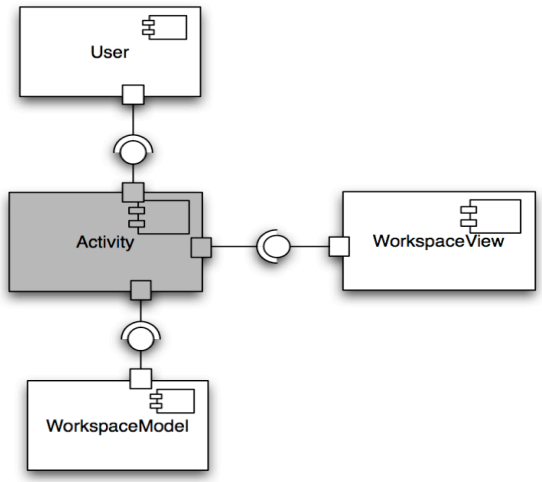


Figure 5. 13: The activity Component in SCOOP interacting with the User, WorkspaceView, WorkspaceModel components

We focus in this research on the synchronous brainstorming and software design activity across multiple-sites. Participants can create formal UML artifacts and informal artifacts such as freehand annotations, scribbling etc.

Contract Description. The contract of the Activity components are about the responsibilities of the activity and their needs. This component support dif-

ferent stages of software design activities as defined in chapter 4.

Pertinent Design Patterns. The Activity components behavior depends on state of the current sub-activity. Therefore we use the State pattern to describe and model this component. As described in chapter 4, the four activities present several dependencies that we modeled using mediator pattern that manage and keep the dependency centralized in one object rather than spread all over the activities objects. The composite pattern serves to describe a conceptual model of the brainstorming activity.

Participants. This component interact with the User component, the WorkspaceView component, and the WorkspaceModel component as shown in Figure 5.13.

Awareness Component requirement and considerations

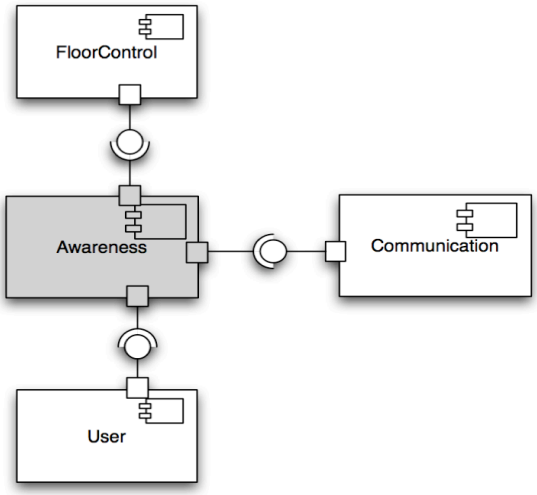


Figure 5. 14: Awareness Component interacting with the FloorControl, User, and Communication components

This component presents an abstraction of collective awareness informa-

tion. In SCOOP we designed and implemented the awareness component in particular for group collaboration in synchronous settings. Therefore, we were concerned with several aspects of awareness information, such as navigation, artifact manipulation, and view representation for group work. Because awareness information is concerned with explicit information presentation, the development of the awareness component had be handled with care. We adopted and modified a research strategy based on a framework developed by Carl Gutwin and Saul Greenberg as illustrated in Figure 5.16. The framework defines a cycle of research and development of awareness information. Our modified approach is described in Figure 5.17. The approach presents a way to identify the different presentation elements of awareness information, see Figure 5.15 for the meta-model (location, activity level, actions, intention, artifacts, abilities, etc....) that are crucial to identify techniques for designing implementing widgets (radar views, peripheral information,...).

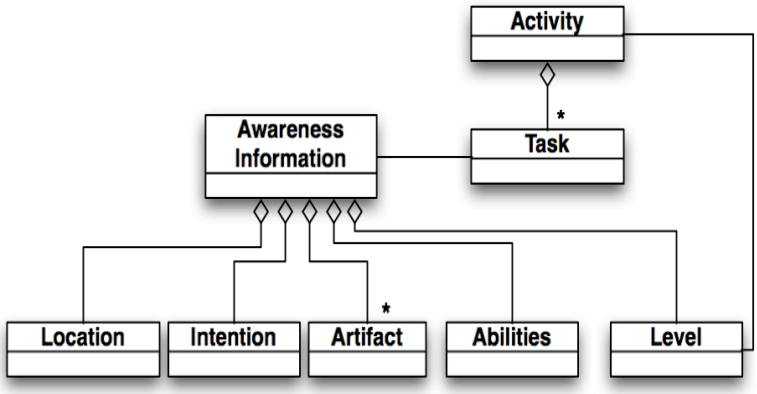


Figure 5. 15: Awareness information in SCOOP

Group workspace awareness component is crucial in the process of guid-

ing users's activities by providing up-to-the-minute knowledge about other person's interactions with the shared workspace and with users. Hence it helps users to understand and coordinate their work [53, 127]. Although SCOOP consider the flow of awareness information from social to, workspace, to peripheral awareness information, only workspace awareness was presented as a reference implementation.

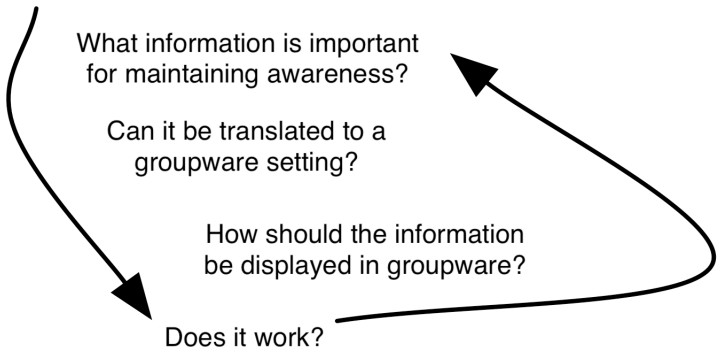


Figure 5. 16: Awareness research framework as described by Greenberg& Gutwin

Contract Description. The contract of the awareness component is about its ability of presenting awareness information and its needs to accomplish its task. Awareness information in SCOOP is presented in several ways. Awareness update its state based on a event-driven mechanism. This component collaborate with floor component to display information about artifacts state (locked, unlocked,...). It collaborates with the user component, to show current user state, his current activity etc. It collaborates with communication component to get notified with messages that were posted by a user.

Pertinent Design Patterns. Awareness component implements the observer pattern. This component registers with the model component to get notified of the model state change and displays the right information accordingly

into the radar view (see chapter 4).

Participants. This component cooperates closely with several components at the same time. the floor control component, communication component, user component, model component, and rationale component.

Invariants. Awareness component role is to convey events resulting from the user action, happening in a sporadic way to the workspace view. Usually awareness information is transient, however, important data that have to be stored is transmitted to group memory, rationale components, and history components that store necessary information. As consequence, Awareness information does not change the state of the model.

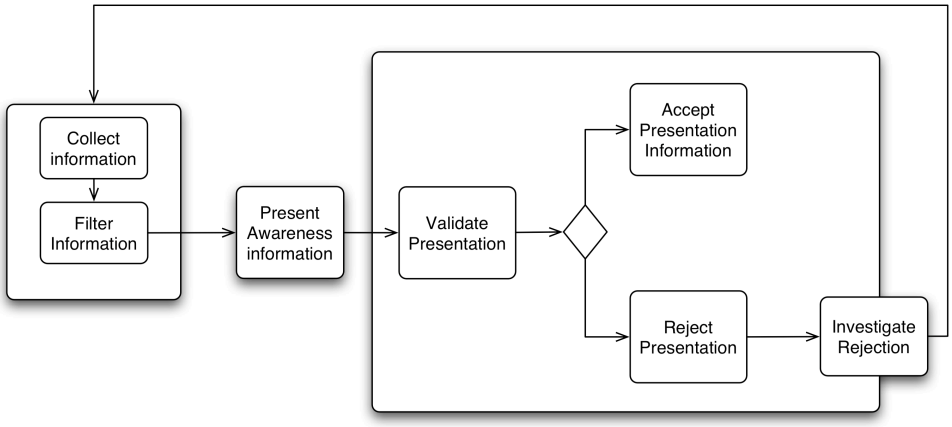


Figure 5. 17: Awareness development cycle adopted and modified from Framework by Greenberg and Gutwin (UML Activity Diagram)

Figure 5.17 depicts the awareness development approach that we adopted to identify important information and their materialization as multiple and integrated views within the workspace.

Communication Component requirement and considerations

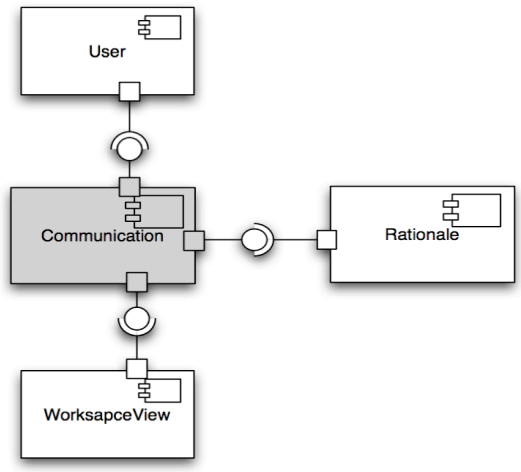


Figure 5. 18: Floor Control Component

The communication component encapsulates both the views and the model behavior of the communication in SCOOP framework.

Contract Description. Communication component role is to enable different forms of instant communication between users of SCOOP to happen. These forms of communication can be switched off and on. We implemented several widgets of communication such as chat (instant messaging), scribbling and communication through artifacts (ArtifactNote class in the analysis model in chapter 4).

Participants. Communication component collaborate with Awareness component, Rationale component, and WorkspaceView component. It sends information about current user to awareness component that in turn displays the right information in views workspace. It transmits related information to the rationale component about which user saying what, about current artifact.

Invariants. Communication component does not influence the model state of the artifacts created.

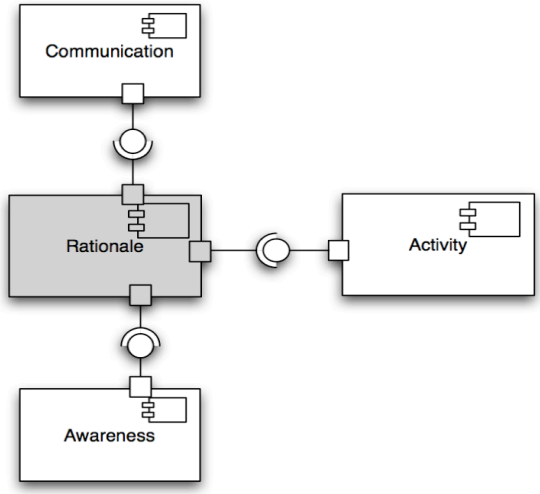


Figure 5. 19: Rationale Component interacting with the Activity component, Communication component, and the Awareness component.

Rationale Component requirement and considerations

This component deals with the rationale information generated during the meeting.

Contract Description. Rationale component encapsulates the rationale model and the rationale views as well. It extracts rationale information from several cooperating components.

Participants. This component collaborates with the Communication component, Awareness component, and the Activity component.

Invariants. Rationale component collects and stores data in its model. It

does not influence the artifacts model.

5.2.Framework Instantiation and reuse

A default or reference behavior of the framework is provided to support distributed modeling activity without having the need to develop any component.

However, any component can be implemented according to the needs of a specific organization and the inter-components interaction is maintained accordingly.

At run-time, new components can be loaded to support a special kind of activities. The same components can be overloaded as well with different possible implementations. The framework allows the switching between one implementation and the other. For example, different implementations for awareness or floor control can be provided.

Awareness information can be as simple as showing a list of the attendee or complex in showing the focus and nimbus information of each user. Floor control can be implemented as explicitly locking an artifact at hand or simply use an optimistic strategy in using awareness information to let users coordinate their tasks.

The interaction between several components (e.g., Activity, awareness, user, floor control…) is set in a central component (i.e., Meeting component) that enables an automatic inter-components communication transparently. This behavior provides a great flexibility in changing the behavior of the application at compile-time and at run-time as well.

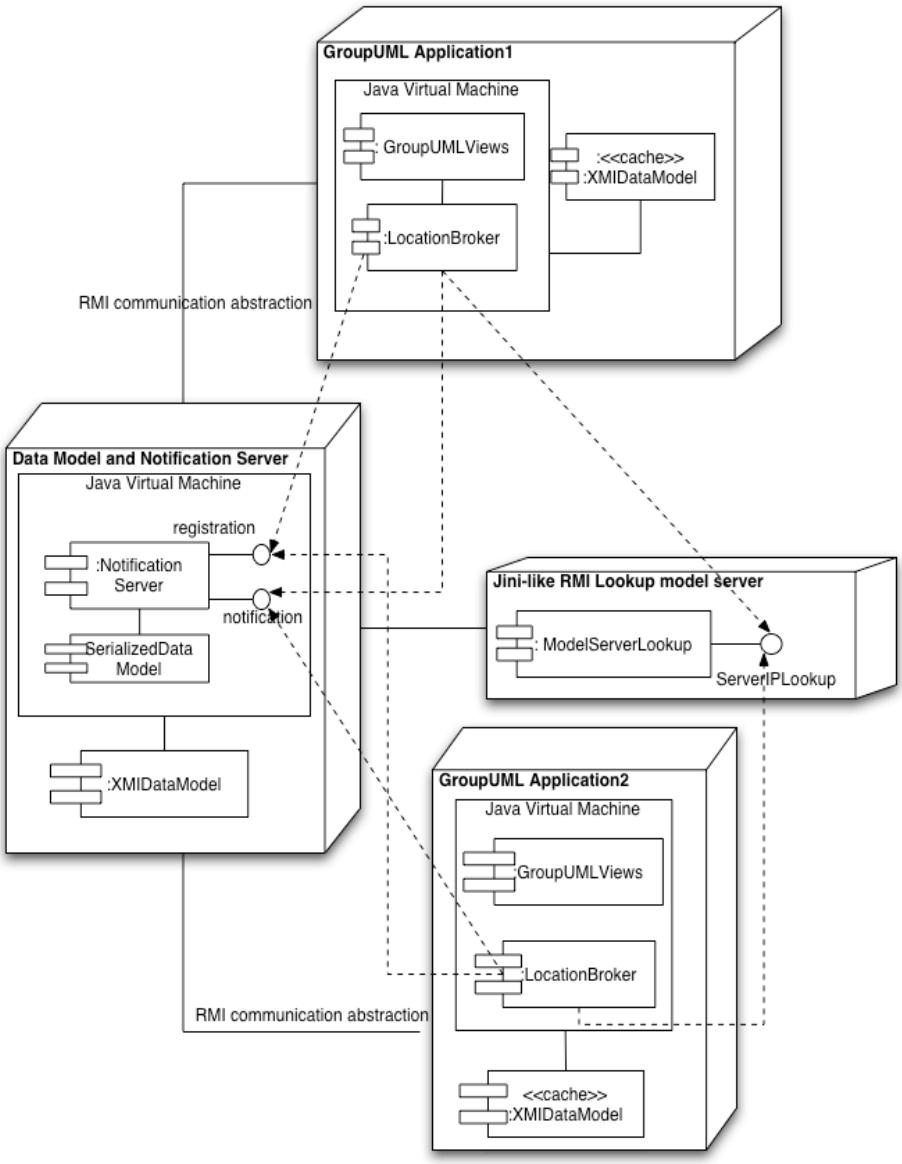


Figure 5. 20: GroupUML deployment diagram

CHAPTER 6

Case Studies

Empirical studies in software engineering are gaining recognition in the software engineering community. Software is not only a matter of computers but is also a matter of humans. Software is developed and used by humans and the human role can no longer be neglected anymore. Software engineering empiricists are considering human aspects now as a major component in software development [19].

In disciplines like social science, qualitative research methods have been developed and are used to deal with the complexities of issues surrounding human behavior [126]. In this chapter we start by introducing qualitative data collection and analysis methods. Next, we show how these qualitative methods are adapted and used in empirical studies in software development. Finally we describe the qualitative methods used in the case studies of this dissertation.

6.1 Quantitative and Qualitative research methods

Quantitative research methods were originally developed in the natural sciences to study natural phenomena. Examples of quantitative methods include survey methods, laboratory experiments, formal methods and numerical methods such as mathematical modeling. They produce statistical results by counting feature, activities, tasks, logging, and measuring certain values of interest in user actions and groups interactions.

Quantitative data are represented numerically or on some other discrete finite scale such as yes/no or true/false.

Quantitative methods are best suited to statistically evaluate a hypothesis that can be translated to a quantifiable value. For example: "groups using a Group Decision Support System perform tasks more quickly than those groups not using it". The time to perform a task can be measured, giving the possibility to support or refute the hypothesis. Quantification may cause the lost of important information. Kaplan and Maxwell argue that the goal of understanding a phenomenon from the point of view of the participants and its particular social and institutional context is largely lost when textual data are quantified [72].

Qualitative research is used to understand, study and explain social and cultural phenomena. It involves the use of qualitative data sources such as surveys, interviews and questionnaires, documents and texts, the observation of participants, and the researcher's impressions and reactions.

Qualitative data is any non-numerical information, represented as words and pictures [65]. *Qualitative analysis methods* are designed to analyze qualitative data such as observations, interviews and diaries. These methods tend to be used when it is necessary to evaluate and understand end user perspectives of a situation.

In contrast to quantitative methods, qualitative methods allow the identification of human-related aspects such as motivation, thinking, attitudes, values, and satisfaction with a product.

The term *case study* has multiple meanings. It can be used to describe a unit of analysis (for example, a case study of a particular organization) or to describe a research method. In this dissertation we are concerned with the use of the case study as a qualitative research method.

6.1.1 Data collection using qualitative methods

Qualitative methods use one or more techniques to collect data. These techniques include interviews, observational techniques such as participant observation and fieldwork. Written data sources include published and internal documents, company reports, memos, letters, reports, E-mail messages, faxes, newspaper articles and so on. The most common techniques in collecting data are participant observation and interviewing.

Participant observation refers to a technique, during which data are systematically and unobtrusively collected." [126]. This does not mean that the observer takes part in the activities. It means only that the observer is visibly present and is collecting data. However, in software development, observation techniques alone are of limited use and should be complemented with other qualitative techniques such as think aloud protocols [8] and field notes [19].

Think aloud protocols require subjects to talk while using the system so the observer can understand the thought process of the participant.

Field notes are written after the observation while listening to or watching the recording of the observation. Then, as soon as possible, the notes are augmented with as many details as the observer can remember. All relevant information contained in the field notes such as place, time, and participants in the meeting, discussions and any other events that took place either as part of the meeting or that impacted the meeting, the tone and mood of the meeting, should be included.

An alternative to this recording method, is to videotape the activities so that the field notes can be collected and written later.

Interviewing is a qualitative data query technique that asks questions to the interviewee. The questions are used to collect opinions or impressions about the activities. They are sometimes used in combination with participant observations where they serve to clarify things that happened or were said during an observation, for example to elicit impressions of a meeting or to collect information on relevant events that were not observed.

There are two types of interviews, *structured* and *unstructured* interviews. In structured interviews the questions are controlled by the interviewer and the response rests with the interviewee; in unstructured interviews the interviewer is the source of both, questions and answers and the interviewee simply has to choose one of these answers [144].

Coding Some researchers [45], [73] suggest to combine quantitative and qualitative to achieve a goal. This process is called *triangulation*. A commonly used technique called coding extracts values for quantitative variables from qualitative data (collected from observations or interviews) in order to perform some type of quantitative or statistical analysis.

6.2 Software developments approaches

Brainstorming and software design meetings in single sites are well known, but geographically distributed meetings are not well understood. As a result the requirements for building software that supports distributed meetings are not clearly defined either. The behavior of participants during same time / different place meetings is not necessarily predictable and we are not able to state clear hypotheses that can be tested and validated experimentally. We, therefore, have chosen to explore and discover the issues surrounding distributed brainstorming meetings, evolve the solutions and investigate their impact on the way meetings are conducted. Thus, as an overall approach, we adopt an exploratory qualitative case study [41].

The lack of clear requirements specification and the lack of development

alternatives leads us to the consideration of an exploratory approach. This is especially applicable, when both customers and developers do not yet have a clear view of their needs. Developers might not be able to make critical design decisions or might not know how to best solve certain implementation problems. By experimenting with prototypes both customers and developers can gain new insights into the problem and may come closer to better solutions.

Instead of exactly planning the various phases of the life cycle, exploratory software development takes small development steps, through which a single step results in an enhancement or extension of the current version of the system. However, the system is not developed based on the prototypes but rather from scratch. The prototypes are used only to improve the developers' understanding of the system requirements [63].

Evolutionary development is an iterative and incremental approach to software development. Rather than creating a comprehensive artifact, such as a requirements specification, that is reviewed and accepted before creating a comprehensive design model, developers evolve the critical development artifacts over time in an iterative and incremental way. The system is released incrementally over time instead of building and then delivering the system in a single release.

In the evolutionary development approach, requirements, modeling, coding, and testing are all continuously evolving together. Evolutionary approaches to software development are supported in agile processes like Extreme Programming (XP), Feature Driven Development (FDD), Dynamic System Development Method (DSDM).

6.2.1 Formative Approach

The formative evaluation approach takes an economical and social science research perspective to assess efforts prior to their completion for the purpose of improving the efforts. It is a method that has become well developed in the edu-

cation and training evaluation literature [123], [130].

The formative approach is used in the context of usability studies, rapid prototyping as well as user interface design approaches. Prototypes are built, based on the current stage of the design, then tested with users, and the results are fed into the next stage of development to improve the design. The formative evaluation aims to guide the evolution of the iterative design of the system by providing rapid feedback to the designers to provide a base for the next decisions [57]. This happens by examining, amongst other things, the strengths and weaknesses and the quality of the prototype system, and by generating understandings about how it could be better implemented. By gradually gaining a better understanding of the system and its requirements. The formative evaluation is particularly relevant to systems whose specification is not clear or not fully defined in advance or is likely to change over time.

In software development, the formative evaluation approach can be used to define the scope of a development project and to identify suitable goals and objectives. Formative evaluations can also be used to pre-test ideas and strategies.

The formative method is used for several purposes. In [87] Ramage presents the formative approach as one of five evaluations methods of Computer-Supported Cooperative Work (CSCW) systems. Grudin [70] states that evaluation of CSCW applications is complex and hard, and requires formative approaches based on the methodologies of social psychology and anthropology. Bannon [82] emphasizes the use of formative evaluation in improving aspects of the design of groupware systems during the design process itself. In particular, case studies of interactive CSCW framework incremental design and development can benefit from formative evaluation approach using a qualitative mode of inquiry [61].

Formative Evaluation Activities Formative evaluation activities include the collection and analysis of data over life cycle of the system development and early feedback of evaluation findings to stakeholders ongoing decision-

making and action. Methods used to collect data and feedback include *open-ended* and *exploratory questionnaires*. The questionnaires are aimed at uncovering the processes by which the prototype takes shape, establishing what has changed from the original design and investigate the relationship between inputs and outcomes. Formative evaluation involves several tasks shown in Figure 6.1.

Several evaluation techniques can be used during formative evaluation, for example, observation, in-depth interviews, surveys, focus groups, and dialogue with participants. Depending on the goals of the formative evaluation, it may emphasize one or more of these techniques. Other methods which might be used according to the evaluated process and situation include stakeholder analysis, concept mapping, nominal group techniques, observational techniques, input-output analysis, questionnaires such as *context-free questionnaires*, and categorization methods such as the *content-analysis method*.

Recent forms of formative evaluation -for example, the mutual catalytic model of formative evaluation outlined recently by Moscoso in [20]- emphasize a more inclusive approach to the involvement of stakeholders, and seek to elicit their participation as collaborators in the evaluation process rather than simply as providers of information. Thus, there is increasing interest in participatory, collaborative, and learning-oriented formative evaluations.

We applied the formative approach to the development of the SCOOP framework through several case studies where we used the outcome of each case study to identify design alternatives for subsequent experiments.

We adapted the formative approach to cover not only testing and improving a sequence of prototypes but also identifying new requirements, testing them with users, and eliciting new ideas. The result of these activities is incorporated in evaluation prototypes, which, in turn, influence the formulation of new experiments in which the newly identified and developed functionalities lead to yet another experiment. In other words, we used SCOOP prototype to design and improve SCOOP itself.

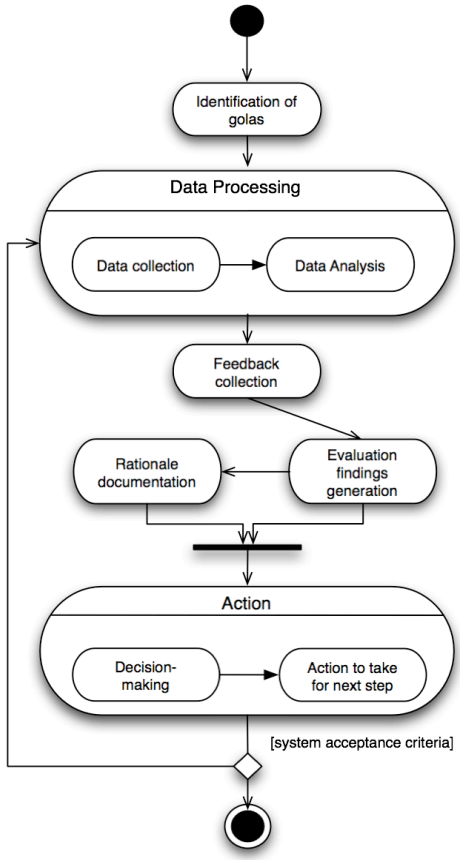


Figure 6.1: Formative approach (UML Activity Diagram): involves the following tasks: the identification of evaluation goals, the planning of data collection and analysis. Then follows a rapid feedback on how the work is going, making value judgments and generating evaluation findings. After, documenting rationale about conflict management and resolution work is proceeding what techniques are used, and what problems encountered. Afterward, planing next step of improving the underlined system, refining goals and data collection and analysis. Finally, executing the plan of making decisions and actions.

The initial requirements for SCOOP were vague and not fully defined for many reasons such as the lack of full knowledge of distributed synchronous

software design meetings and of the potential users and their needs.

The **Context-Free questionnaire** requires the researchers to listen before attempting to invent or describe a potential solution.The goal is not to influence the users feedback, the initial questions are about the nature of user's problem without context for a potential solution. This approach was suggested by Gause and Weinberg in [140].

The **opinion-type** questions ask the interviewee what they think about a certain problem. **Attitude-type** questions ask the interviewee what their attitudes are when working with a particular product.

The **content analysis method** is a research method that uses a set of categorization procedures for making valid and replicable inferences from data (text or images) to their context [140]. This method combines qualitative defining the categories and quantitative aspects, namely determining numbers within categories. For example, answers are categorized into different types, and the numbers of each type are added.

For our case studies we use the content analysis method to categorize collaboration aspects and issues, faced by the subjects and assess their relevance relative to the current step of the software development.

The following section describes our case studies and the empirical experiments.

6.3 Case studies: purpose and approach

Our hypothesis for the case studies was: “Using GroupUML improves distributed synchronous software design and makes requirements engineering in the target development environment possible”. This can be evaluated qualitatively. For our case studies we use techniques of *participants observations*, *interviewing* and *coding*.

The main goal of the case studies was to demonstrate that our research hypothesis holds, that is, that distributed synchronous software brainstorming and design meetings using SCOOP are feasible.

The second goal was to validate the SCOOP framework, and to prove its viability and usefulness for the domain of software systems that require distributed meeting management.

The third goal was to use SCOOP to identify the issues of distributed software design, classify them into categories, and to discover requirements for SCOOP. The categorized issues are then mapped into requirements.

The fourth goal was show the ability of SCOOP to deal with ill-defined requirements making distributed requirements engineering and design in the target environment possible, where developers and users discover requirement, and cooperate incrementally improving SCOOP using SCOOP itself.

Throughout the case studies, we followed a formative approach, not only as an evaluation method, but as a process of designing and developing cooperative software for distributed synchronous software brainstorming and design.

The evaluation methodology involves iterative cycles in which a SCOOP prototype is used, new requirements are discovered and analyzed, revised designs are put forward and the SCOOP prototype is further developed. The prototype is then assessed to refine the requirements and design alternatives.

Through iterative deployment of the prototype in the target environment, participants responses to the use of the prototype are analyzed through observations, questionnaires, and group discussions. By including the participants in the review of the development of SCOOP, the design and the implementation can reflect more accurately the participants ideas, as well as getting more usable feedback. The review process relies mainly on qualitative methods to gather issues, analyze requirements, and evaluate alternative solutions continuously.

The case studies were conducted in an environment described in section

3.1 and can be categorized as a set of three main issues: communication and coordination, awareness and control, and rationale management.

Communication and coordination issues

In distributed meetings, communication takes place explicitly, by means of formal such as using phone or chat tools, or by informal means of manipulating shared artifacts. The framework must provide communication means such as chat or multi-cast audio. In the first case study, we discuss the set of communication and coordination issues the participants encountered during the initial meetings. We discuss the functionalities implemented, such as instant messaging, live annotations, and the lock mechanisms.

Awareness and control issues

In this study, we brainstorm requirements to deal with awareness and control issues such as the capability to identify individual users and track their status; their actions, and their history. We describe the support several levels of awareness information such as informal awareness and workspace awareness. Informal awareness provides information about local and remote meeting participants (such as user picture and name, status active or inactive) to guide their work. Workspace awareness provides up-to-the-minute knowledge about other person's interactions with the shared workspace and with users (such as current user focus, activity, and task) to help them coordinate their work. We evaluate the functionalities implemented, such as radar view, interactive user awareness list view, ticker, and fader. Finally, we assess the status of the prototype and its evolution.

Rationale knowledge and memory issues

In this case study, we describe the set of rationale knowledge and memory issues participants encountered during the meetings. The challenges that have been considered such as:

- designers have to be able to express their design reasoning in a natural way,

as it is generated during design meetings

- the representation must be formal enough to be processed for classification, browsing, and enabling search methods.

Finally we describe and the functionalities implemented (such as the QOC assessment matrix).

6.3.1 Experimental Context

The case studies that took place within the university program for post-graduate participants and Master students of the computer science curriculum in the Technische Universität München. We conducted the experiments over a duration of four semesters with a total of seven groups averaging six participants each see (Table 6.1). A full cycle of development lasted a semester long, during which a series of experiments were conducted consisting of a regular series of distributed meetings to design and develop cooperatively the SCOOP framework.

During each semester we had two separate groups of six participants each[1]. This enabled us to conduct two independent series of meetings running simultaneously. Each group was split into two sub-groups of three users each. During each meeting, each sub-group of participants was provided a different room and the necessary hardware and software to run the meetings.

The hardware consisted of Smart Boards [133] connected to networked computers via the internet (see Figure 6.4). Each computer ran the application prototype GroupUML [103] enabling different groups in different locations to collaborate over system architectural design using UML. A short tutorial about how to use GroupUML and the Smart Board was provided at the beginning of each development cycle.

1.with the exception of the summer semester 2004 where only one group was conducting the experiment

Semester	Year	Groups	Size	total
Winter	2003	2 groups	2 x 7	14
Summer	2003	2 groups	2 x 8	16
Winter	2004	2 groups	2 x 5	10
Summer	2004	1 group	1 x 5	5
				45

Table 6.1: Subjects participating in the case studies during the four semesters.

In the first development cycle (see Figure 6.2 for the distinction between case study, iteration, and development cycle), a minimal basic architecture was built to support collaborative brainstorming and sharing of graphical representations of UML artifacts. This step involved only the researcher of this work.

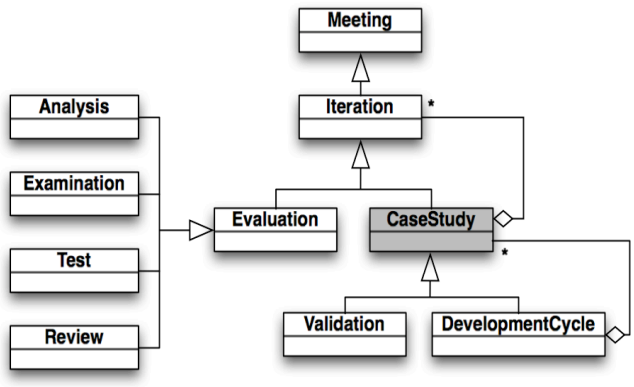


Figure 6.2: Case study structure: a one semester development cycle consists of a set of case studies. A case study can take several iterations (meetings) to complete.

In each subsequent development cycle, a prototype was incrementally developed, tested and deployed in the target environment. Current features were

used to identify and design missing features. Features of the prototype were then added or removed according to the requirements. The prototypes evolved dynamically till getting closer to a final version that fulfilled a given requirement. The SCOOP framework evolved also in an incremental way where associated components designed were added or modified. Therefore, the prototypes and the framework evolved incrementally together over time.

Problem definition (Case study I)

Along the semester you will brainstorm requirements and design a model for distributed application enabling synchronous editing of graphical artifacts. The system should help people in different locations to share and discuss ideas using UML and scribbling. For the moment skip implementation issues.

For this task use GroupUML and collaborate with your colleagues located in room2.

Establish the following functionalities:

1- communication with your colleagues

2- sharing artifacts

3- accessing artifacts

4- storing artifacts

Hint: try to model your current activity and your needs while you are brainstorming with colleagues.

Figure 6.3: Initial requirements given to participants

6.3.2 Experimental Case Study I: communication and coordination issues

Based on the initial version of GroupUML, participants were provided with the problem (see Figure 6.3) to improve distributed collaborative design utilizing GroupUML itself. The design activities varied from brainstorming to system modeling.

The participants were actually put in the same room but with two Smart Boards that were put back to back in a way that each group of participants faced one Smart Board but could not see the other group (see Figure 6.5).

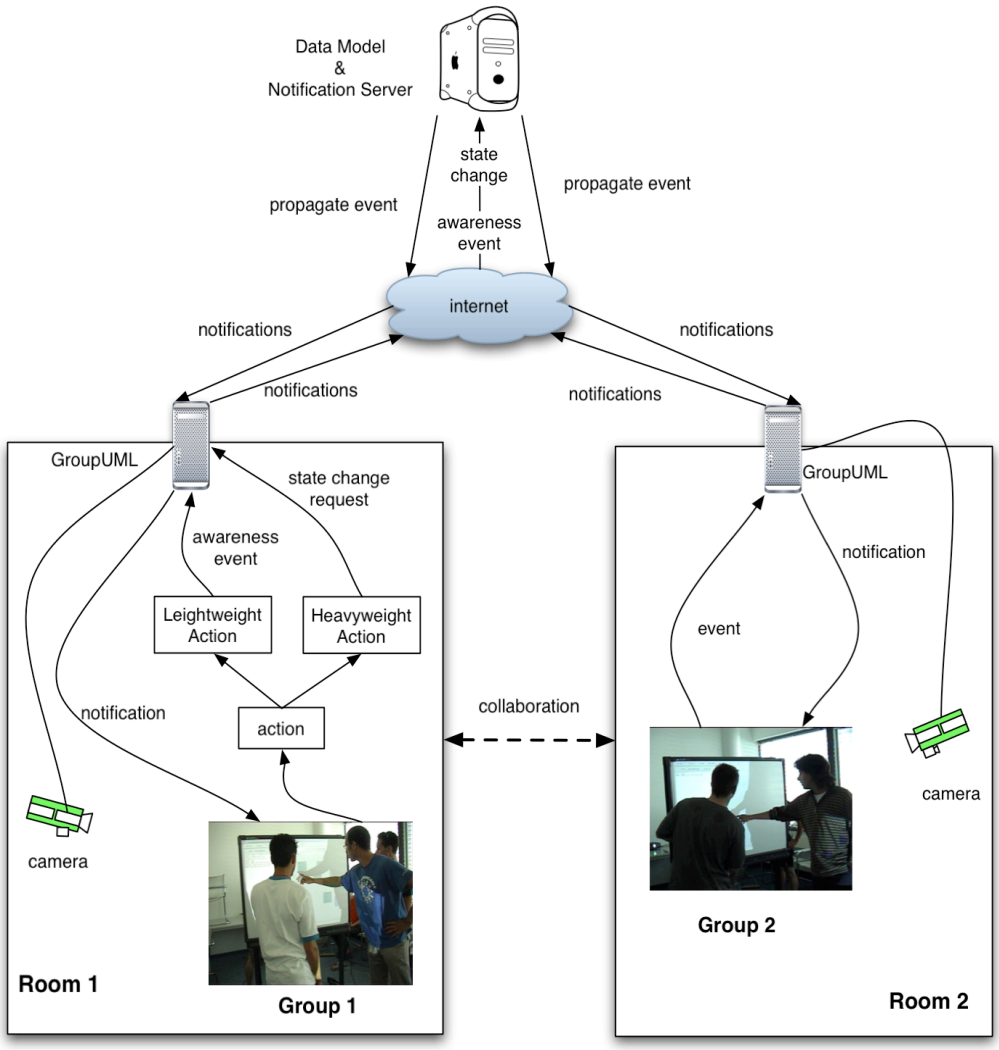


Figure 6.4: Experimental setup for distributed same time / different place software design and brainstorming meetings: two different groups located in two different rooms, are videotaped, and collaborating over software design using GroupUML. The remote users use Smart boards to interact with GroupUML and their peers. GroupUML updates the model located in a remote server through remote notifications. The model server propagates the change to all GroupUML applications.

While we (the researchers of this work) could observe both groups simultaneously, the two groups were not allowed to speak directly to each other, only GroupUML was allowed for communication and collaboration.

The participants were videotaped during the meetings and surveyed with a questionnaire afterwards. The answers to the questionnaire served as a basis for a group meeting in which encountered issues were discussed.

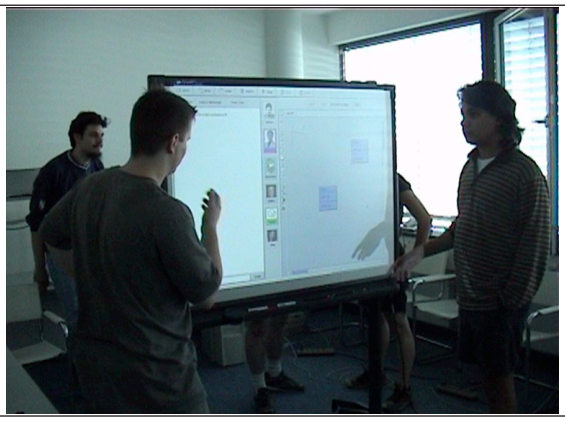


Figure 6.5: Single room meeting: in the initial meeting, participants were in the same room but can't communicate only using GroupUML

Several iterations, that is, meetings, are needed to solve an issue. The purpose of each iteration was to identify issues, propose solutions, develop, deploy, and valid functionalities, then move to the next iteration.

During this first iteration, participants had neither floor control mechanism nor communication means so that they could experience the difficulty in coordinating their actions in a distributed setting. The only views available during this case study were the UMLView and ScribbleView.

The participants had little knowledge about SCOOP requirements. During the initial phase of discovering ideas and brainstorming, one participant from one side tried to rename a UML class at the same time another participant from the other side pointed to the same class and moved it away. As a result, the action of the first participant was canceled and he had to try again.

The participants faced such a incident several times and tried to solve the problem their way using the scribbling to highlight the artifact under consideration although it was not defined for such a purpose. Another solution they tried was to duplicate classes and then decide which one to keep. In the end, participants were asked to fill in a questionnaire where they were guided to express their impression and their views and suggestions of what they have just experienced.

We provided a context-free questionnaire composed of o*pinion-type* questions and *attitude -type* questions. Figure 6.6 shows the questionnaire given to the participants.

Questionnaire (Case study I)

Opinion -type questions

- What does GroupUML enable you to do?
- What do you like/dislike in this groupware?
- What do you expect from GroupUML?
- What's missing according to you?
- What do you suggest to make GroupUML better support users?

Attitude-type questions

- Do you feel being efficient when working with GroupUML for collaboration?
- To which degree you like the GroupUML?
- How helpful do you feel GroupUML is in collaboration?
- To what extent do you feel in control of the interactions with GroupUML?
- Do you feel you can learn more about GroupUML by using it?
- Did you manage to solve the issues under investigation?
- Could you take profit from the collaboration with your peers

Figure 6.6: Initial context-free questionnaire composed of opinion-type and attitude-type questions.

Results and interpretation (Case study I)

We noticed that with small groups, case studies are easier to manage and control. It was also interesting to observe that the participants organized themselves: two to three participants interacted with the system using the Smart board (to create, move, and select graphical artifacts) and one participant used the keyboard to enter text.

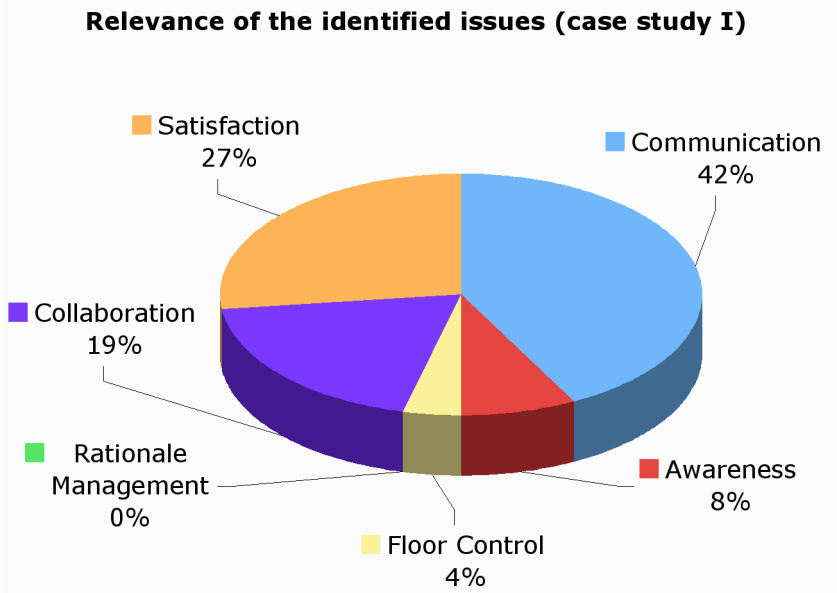


Figure 6.7: Relevance of issues in the initial meeting: communication issues were most important to the users and consequently were targeted for the second meeting.

Figure 6.7 represents the relevance of issues from the viewpoint of the participants. These results form the basis for our second iteration. For example, communication issues were considered most important, thus, for the second iteration we emphasized the support for communication means.

From analysis of the responses to the questionnaires, we conjecture that participants enjoyed using such a system even if it lacks many features that they would expect to see. While the participants were aware of the presence of their colleagues, most of them emphasized communication issues and synchroniza-

tion issues. Others expressed a need for a mechanism that enables concurrent access to artifacts. One of the participants expressed a wish to see a lock and unlock feature to enable the control of flow of events.

During subsequent meetings, participants were asked to brainstorm and design models that solves these issues.

Several use cases were designed to describe the locking functionality (Figure 6.8). In addition, the UML class model of GroupUML was designed and refined to describe the lock functionality according to the select strategy context (Figure 6.9).

In the second meeting of the initial case study, two functionalities were rapidly built into GroupUML: a chat feature to enable communication and a lock/unlock mechanism for explicit synchronization of participants' actions.

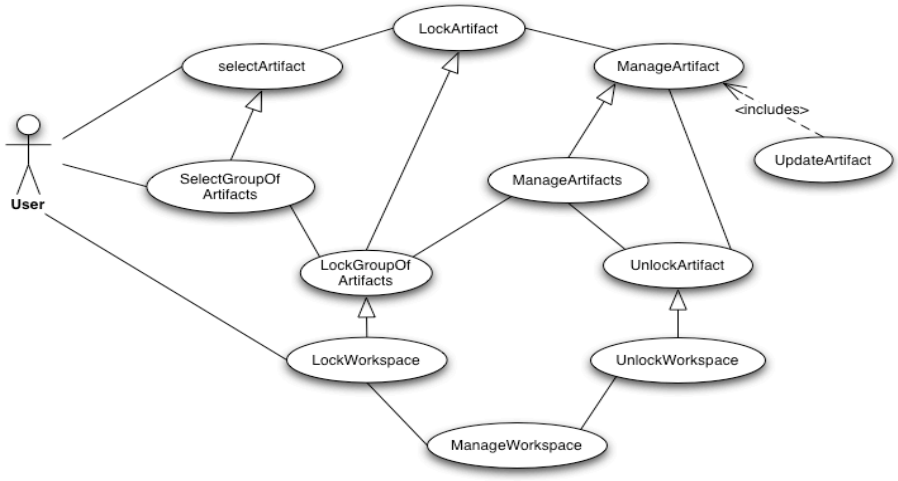


Figure 6.8: Lock use cases built during the case studies

Figure 6.10 depicts the user interface of the lock mechanism which shows a locked workspace (selection palette is greyed) for the current user and a red ticker that gives a hint about who is currently owns the lock. The hint enables users to ask explicitly the one who owns the lock, to unlock it.

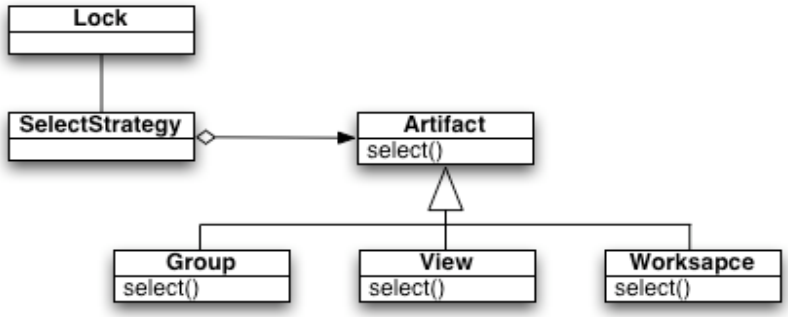


Figure 6.9: Lock model designed during the case studies

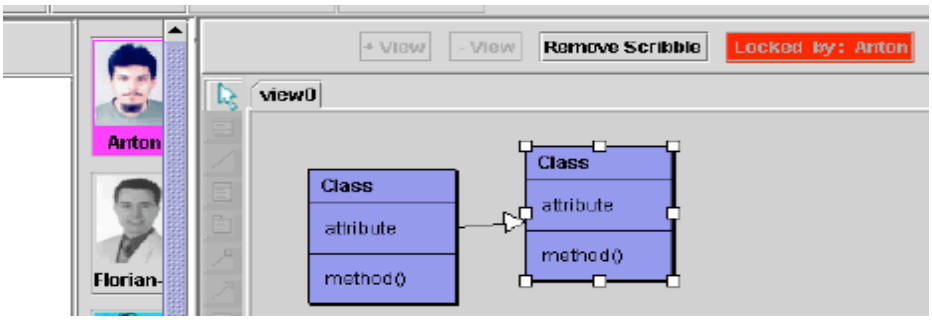


Figure 6.10: The user interface for the lock mechanism: a red ticker showing who is currently locking the workspace.

According to the selection strategy, selected artifacts or views, can be locked by the user owning the floor. If nothing is selected, the user can lock the whole workspace.

Lessons learned (Case study I)

The case study consisted of three iterations. In the second iteration, participants were shown the newly added features and were asked to keep on carrying their tasks with the new features.

The chat feature was straightforward but not the lock/unlock feature a par-

ticipant on one side can lock the workspace until he is ready with his modification and then unlock the workspace. At that moment, participants from the other side can only wait and use the instant messaging to ask the other side to unlock the workspace. This new situation generated new constraints and criticism of participants. A refinement of the solution was suggested that involves the locking of the artifact under access only and not the whole workspace.

In the third iteration, participants tested and evaluated the refined solutions. This iteration ended with the review and validation of the new functionally

The participants were using GroupUML to collaborate over designing the GroupUML itself. They were involved in suggesting solutions to issues, that they not only know theoretically but also faced in the reality while using the tool. They designed initial solutions to problems that were refined and implemented. Using the improved groupware with more functionality. The participants were using the features over several meetings and faced less difficulties, a sign for the acceptance of the solutions the participants came up with and the emerging maturity of the groupware.

6.3.3 Case Study II: Awareness issues

The participants of the second case study had some background knowledge of awareness but did not yet realize what kind of concrete information they needed to see on the Smart Board to support distributed real-time meetings. The participants noticed that if they receive no instant messages from the other side, they just keep still and observe the views on the Smart Board. They had no idea what the others are doing and why are they silent. Discussing about these moments of silence they realized that they need more information about their remote peers. The questionnaire (see Figure 6.11) was designed to establish what kind of levels of information they wished to see on the Smart Board to coordinate their actions and avoid the “silence periods”.

In the second iteration, participants were asked to brainstorm (see Figure 6.12) and design awareness models that solves the awareness issues they faced in the experiment and sketch possible implementations.

Questionnaire (Case Study II)

Communication

- Do the communication means of GroupUML enable you to collaborate with your remote colleagues in conducting brainstorming and modeling tasks?
- Do you prefer other means of communication you think they improve collaboration? please enumerate them.

Awareness

- Do you get enough awareness information provided by GroupUML?
- Which level of awareness information you need in conducting your task (user, activity, atomic task, and so on)?
- Do awareness information enable you coordinate your task with remote colleagues?

Floor control

What do you think about the following ways to deal with floor control issues (who has the turn to talk, communicate, draw, access.):

- Implicit (without locking): rely on your social behavior
- Explicit (lock button): locking workspace, group of artifact, an artifact
- Any other suggestion to these issues?

Rationale management

- Do you know "the why" behind design models you created in previous meetings?
- Do you capture the design decisions and the reasons behind them?
- What mechanisms you need to support you dealing with knowledge and history information?

Synchronous behavior

Does GroupUML enable you to handle real-time interaction with your peers. What do you think about the responsiveness?

Functionalities

What functionalities are needed according to you to improve the collaboration process with GroupUML? please give your suggestions and opinion.

Figure 6.11: Subsequent questionnaire given to students in Case study II

Results and interpretation (Case Study II)

The analysis of the answers is summed up in Table A.1 (see appendix A). Most of the participants requested to get more awareness information about the current task, a few of them mentioned presence awareness. As a whole, aware-

Problem definition (Case Study II)

You have seen many different ways of dealing with awareness as well as floor control in distributed settings. The goal of this meeting is to identify or improve current support of these functionalities to GroupUML.

Task:

Use GroupUML to express your ideas. Collaborate with your local and remote team-members to:

1- Brainstorm and design an awareness system support to GroupUML

2- Brainstorm and design a floor control system support to GroupUML

Figure 6.12: Awareness and floor control requirements given to the participants (Case study II)

ness issues were more apparent. Mostly because participants used the improved functionalities of GroupUML in communication and floor control management but had little support of awareness information.

Figure 6.13 shows the relevance of issues of the current experiment. Participants are now paying attention to issues that were not yet addressed completely. Awareness and rationale management are gaining more attention from the participants. Most of communication issues were solved as well as floor control issues, as can clearly be seen in Table A.4.

Two solutions were retained and implemented for the third iteration. Several use cases were designed to describe the awareness functionality (Figure 6.14). In another step, an initial UML class model was designed and refined to describe the possible awareness functionalities according to the provided selection strategy context (Figure 6.15).

As a result, two awareness concepts were immediately built into the GroupUML: First, user awareness functionality to enable propagation of user presence across the meeting groups (see Figure 6.16). Second, radar view that shows a miniature of the desktop, overlapped with current focus of the local and remote participants (Figure 6.17).

Interactive user awareness functionality: Before starting any action, every participant must click his icon. The icon is then highlighted in green (for

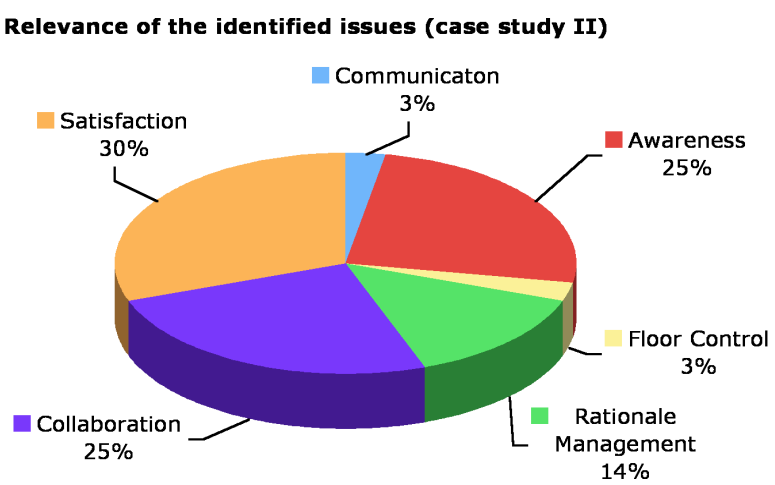


Figure 6.13: Relevance of issues according to current experiment

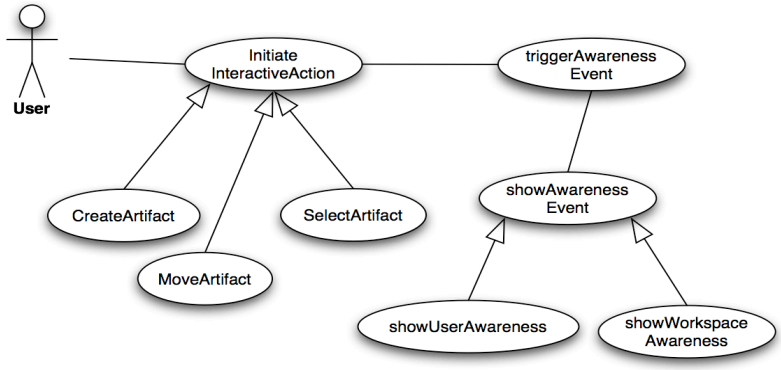


Figure 6.14: Awareness use case model designed during case studies: according to the user action, awareness information events are triggered.

local user) in magenta (for remote user). In one site there is always one active icon (highlighted in green) and one or several icons are highlighted in magenta for the remote user (see Figure 6.16). This feature is also used to document the group history and the design rationale.

According to some participants, knowing who is currently working on the

board is of less importance then knowing the current action. Whereas other participants argue that knowing the current user is of importance since they may be able to anticipate his future actions.

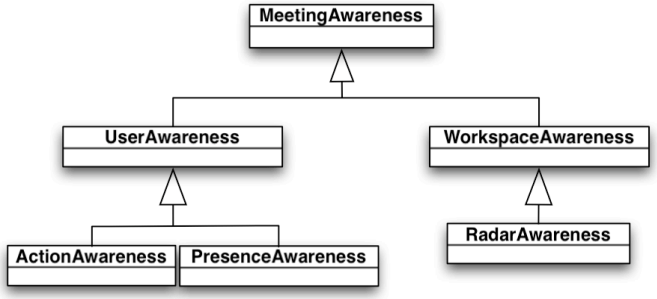


Figure 6.15: Initial awareness model designed during case studies

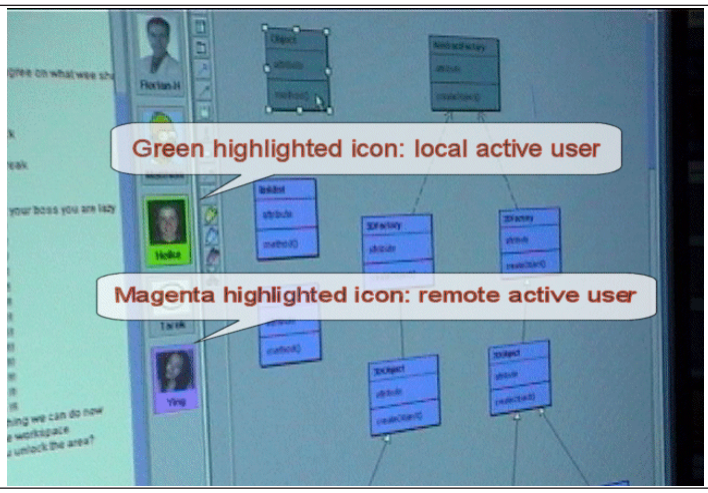


Figure 6.16: Group awareness: Magenta color used for remote user and the green for local user

Radar feature: This is an implementation of the Nimbus/Focus model. It shows in real time the current user workspace area of focus. If the user scrolls to a direction, the corresponding radar is updated so other users are aware of his or her current focus (Figure 6.17: blue area, at the lower left corner of the radar tab,

shows where the remote site group focus is, and the gray one, at the upper right corner of the radar tab, shows the current area of focus of the local site).

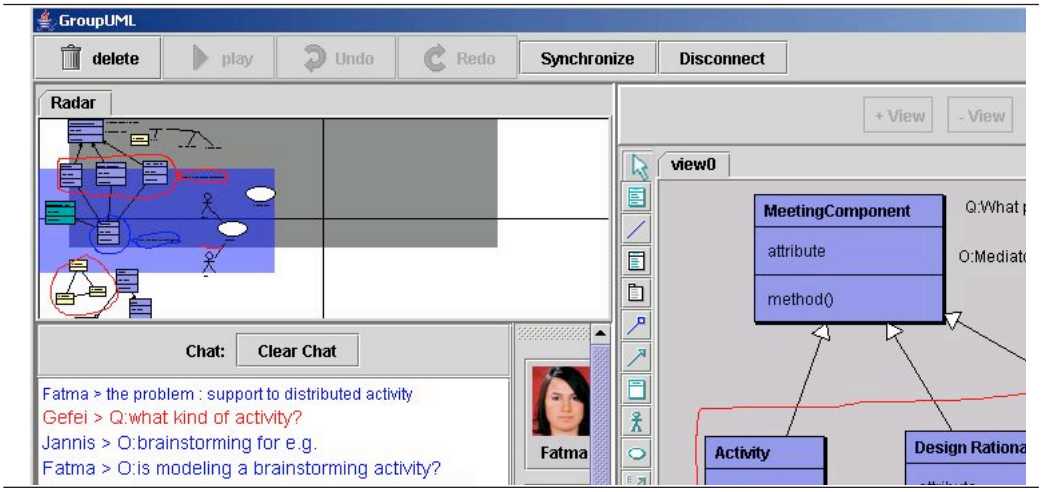


Figure 6.17: workspace awareness - Radar view -

Lessons learned (Case Study II)

The initial idea behind the radar feature was that a participant highlight an object and ask the remote user if it can be deleted. Without the radar feature users send messages to the other side, but if the remote participants have scrolled to another area of the workspace they didn't realize that they don't look to the same view. Consequently, participants were asked to give their suggestions to solve this issue and to mention what kind of category of issue it belongs to.

Most of the participants suggested synchronizing the scrolling of all views, but then they realized that this could disturb a user currently working on an artifact in a different view. Finally, a solution was adopted to have a global view of the workspace and the focuses of each site.

This feature made collaboration considerably easier. On the other hand, the interactive user list provides some drawbacks: if a participant forgets to click the icon, the previous participant will be considered as the initiator of the cur-

rent action. This is can be an issue in case of a distributed meeting where roles are of significant importance. The main drawback of the radar feature is having multiple views (i.e.; tabbed views e.g., each representing an alternative): First, it is hard to overlap many focusing areas (we use different colors to show the focus of different views); second, it is hard to determine which area radar belongs to which view in a given workspace.

Finally, several awareness features were experimented with and implemented in GroupUML such as ticker and fader interfaces.

6.3.4 Case Study III: rationale knowledge and memory issues

Although design rationale is of great importance, it is very often neglected [10]. One way to make participants aware of its importance, is to challenge them to answer questions such as “why did you decide for this design?” Or “why use this pattern?”. The answer is often that they did not remember their decisions made two months ago and even two weeks. The questions aim to let participants think and feel that they need a mechanism to document their decisions, the alternatives and the options they considered.

Also they were encouraged to think about adopting rationale management as an integral part of distributed meetings, and not as an optional activity. To do this, several issues were presented to the participants in the form of three tasks:

- Define rationale management activity for distributed meetings
- Determine the deliverable of the rationale management activity
- Link rationale information to brainstorming artifacts

Results and interpretation (Case Study III)

After several brainstorming meetings and discussion of the different alternatives to solve these problems, participants were asked to fill out a guided questionnaire.

Figure 6.18 shows the relevance of issues of the current experiment. One can see that participants are paying attention to rationale issues that were not yet addressed completely.

Two preliminary design models were produced, rationale use case model (see Figure 6.19), and a rationale class diagram model (see Figure 6.20). These models were used as a starting point to design the rationale component of the SCOOP framework (see chapter 5, SCOOP subsystem decomposition).

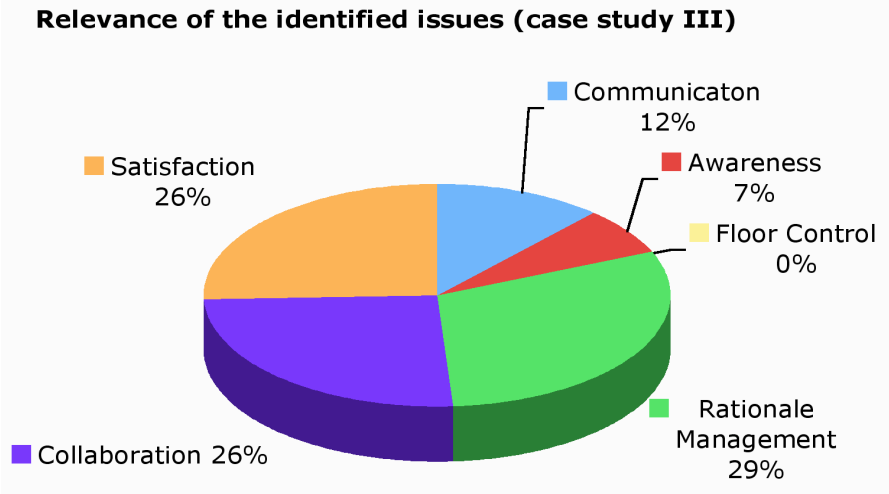


Figure 6.18: Relevance of issues according to current experiment

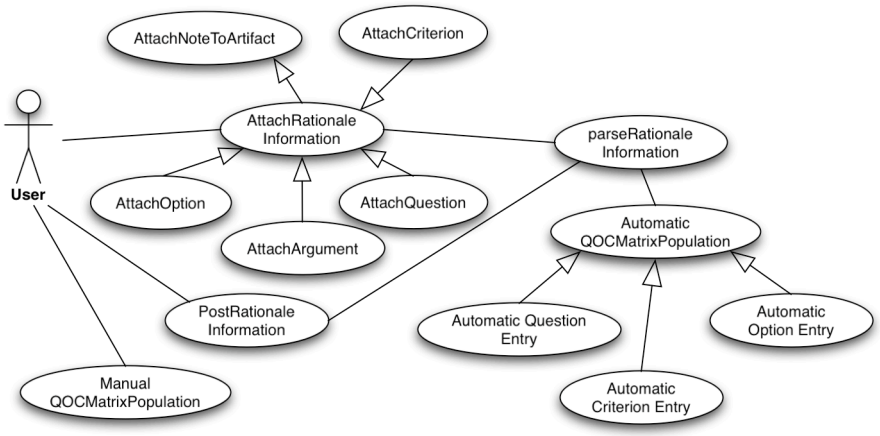


Figure 6.19: Rationale use cases model designed during case studies

Participants approached different ways to support rationale. For example, some suggested to force the user to fill in a text area with all steps that better they can perform another action (such as the creation or deletion of an artifact). This was against the principle of motivating users to integrate rationale management in groupware and at the same time reducing the overhead in capturing rationale by not requiring any substantial change in the designer's activities.

The different ways considered to integrate and manage rationale within distributing brainstorming activities are discussed below:

- **Manually rationale information entry**

Users enter rationale information about current artifact, group of artifacts, or view in an assessment table that is used to store rationale information and to manage conflicts during the *consolidation* and *conflict resolution* phases (see Figure 6.21).

Attaching a question, an option, or criteria to an artifact, so every artifact in the workspace, being a formal (UML) or informal (other), represents not only the artifact itself but also its history and its raison d'être. Participants select the kind of link they like to attach to an artifact, and then point to the artifact and type in a question, an option, or a suggestion. The benefit of this feature is to create self-contained artifacts and at the same time, sharing the attached questions and suggestion with other users, (see Figure 6.23).

Arranging the project under development, the issues identified, the options that are represented through diagrams, as a tree-like representation (see Figure 6.22). Users can point to the project and with a double-click a question container is created with the corresponding sub-options inserted. Each option is pointing to a diagram or set of diagrams that shows the different alternatives that were considered. The advantage of this representation is that participants can have a quick view of the issues and their options at a glance.

- **Semi-automatic rationale information entry**

Users can switch on this feature (described below) to trigger a semi-automatic entry of different rationale information from three different resources:

A simple presentation of the rationale as a QOC (Question, Option, Crite-

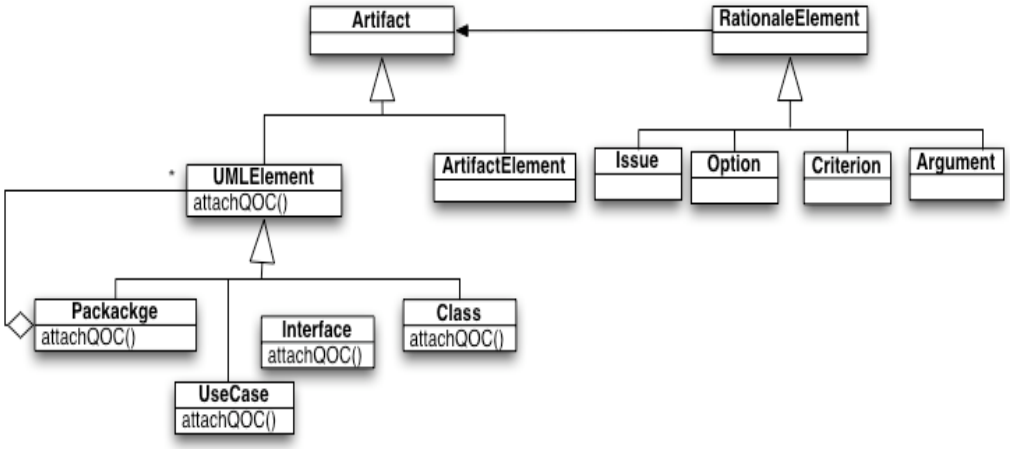


Figure 6.20: Initial rationale model designed during case studies

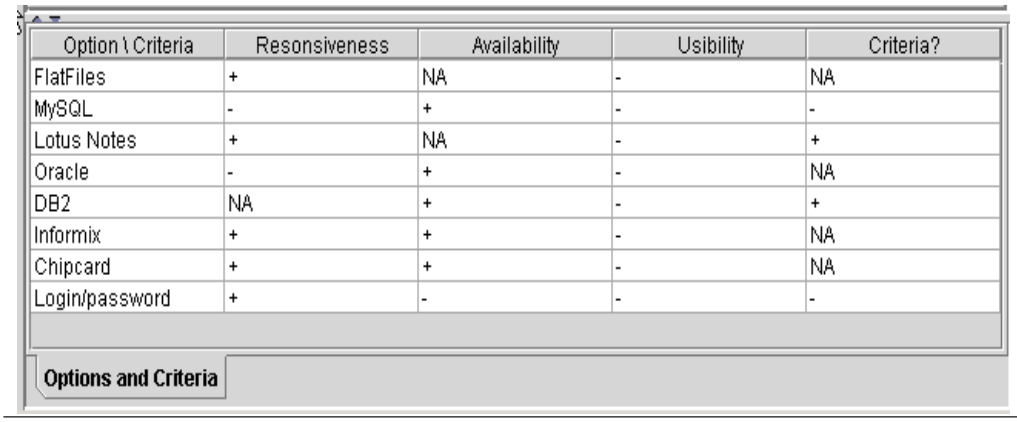
ria) matrix for evaluation of the different alternatives and conflict resolution. Each question/option/criteria attached to an artifact by participants, is automatically inserted into the QOC matrix. This shows a clear and concise presentation of the issues discussed and the option chosen. Participants need only to enter a "+" or "-"to evaluate an option against a criteria. GroupUML then computes the final choice of alternatives automatically.

Users post messages to each other using the instant messaging view, that contains rationale information (see Figure 6.24). When posting a question that concerns a current view or currently selected artifacts, users need to mark a check box mentioning that the message is about a question, an option, or a criterion. Messages are parsed and automatically inserted into the QOC Matrix.

Users can enter text description about artifacts directly into the workspace area. Text placed on the workspace is analyzed and parsed periodically. Every identified rationale information is entered automatically into the matrix.

We conducted three iterations with these rationale methods, and observed that there is a need for a meeting role that cleanups the workspace, and manually enters rationale information, that was not detected automatically by the sys-

tem, manually into the assessment matrix.



Option \ Criteria	Resonsiveness	Availability	Usibility	Criteria?
FlatFiles	+	NA	-	NA
MySQL	-	+	-	-
Lotus Notes	+	NA	-	+
Oracle	-	+	-	NA
DB2	NA	+	-	+
Informix	+	+	-	NA
Chipcard	+	+	-	NA
Login/password	+	-	-	-

Options and Criteria

Figure 6.21: Assessment table: evaluating options against criteria

Lessons learned (Case Study III)

The rationale aspects were not obvious for participants to assimilate, although they were aware of the short-term and long-term benefits of the rationale.

At first glance, participants were not able to imagine the integration of rationale as seen during the tutorials and discussions into their meeting activities. Some of them though that the documents such as SDD (System Document Design) are sufficient to document rationale but then they realized it's not possible to put each alternative and each decision into the same document. Participants understood mapping the rationale into a QOC matrix and contributed with some visionary-scenarios of managing rationale. Use cases and class diagram of rationale models were then developed jointly by participants and improved to fit into a rationale component within the SCOOP framework.

Rationale should not be considered as an independent or optional activity but should be part of all brainstorming activities and developers have to be concerned with it from start. GroupUML was used to model its support to rationale using previously developed functionalities such as scribbling, communication,

awareness information. Associated models generated by GroupUML were developed and integrated within GroupUML, tested and validated and finally adopted as a main component of SCOOP framework.

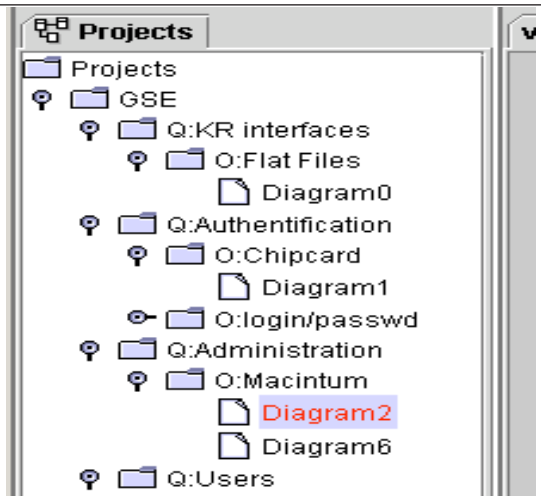


Figure 6.22: QOC tree representation: for a project, questions have one or several diagrams associated to them. This gives a quick view of the different issues and their related solutions

The GroupUML user interface with the features lock mechanism, awareness, communications, and rationale management views is shown in Figure 6.24

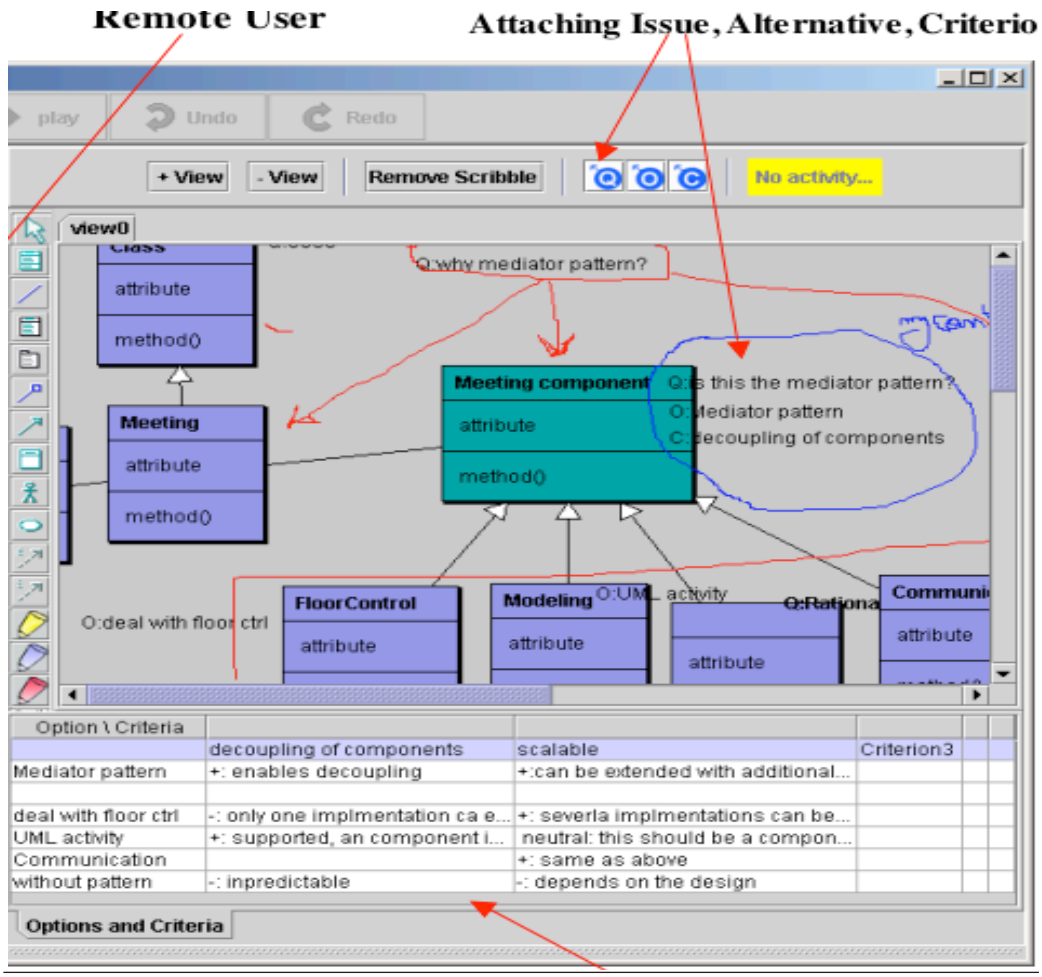


Figure 6.23: Attaching QOC elements to artifacts

6.4 Results and Discussion

In this chapter, we have presented a formative empirical method for conducting experiments of distributed synchronous collaborative software design and brainstorming meetings.

A set of case studies were conducted during four semesters involving a

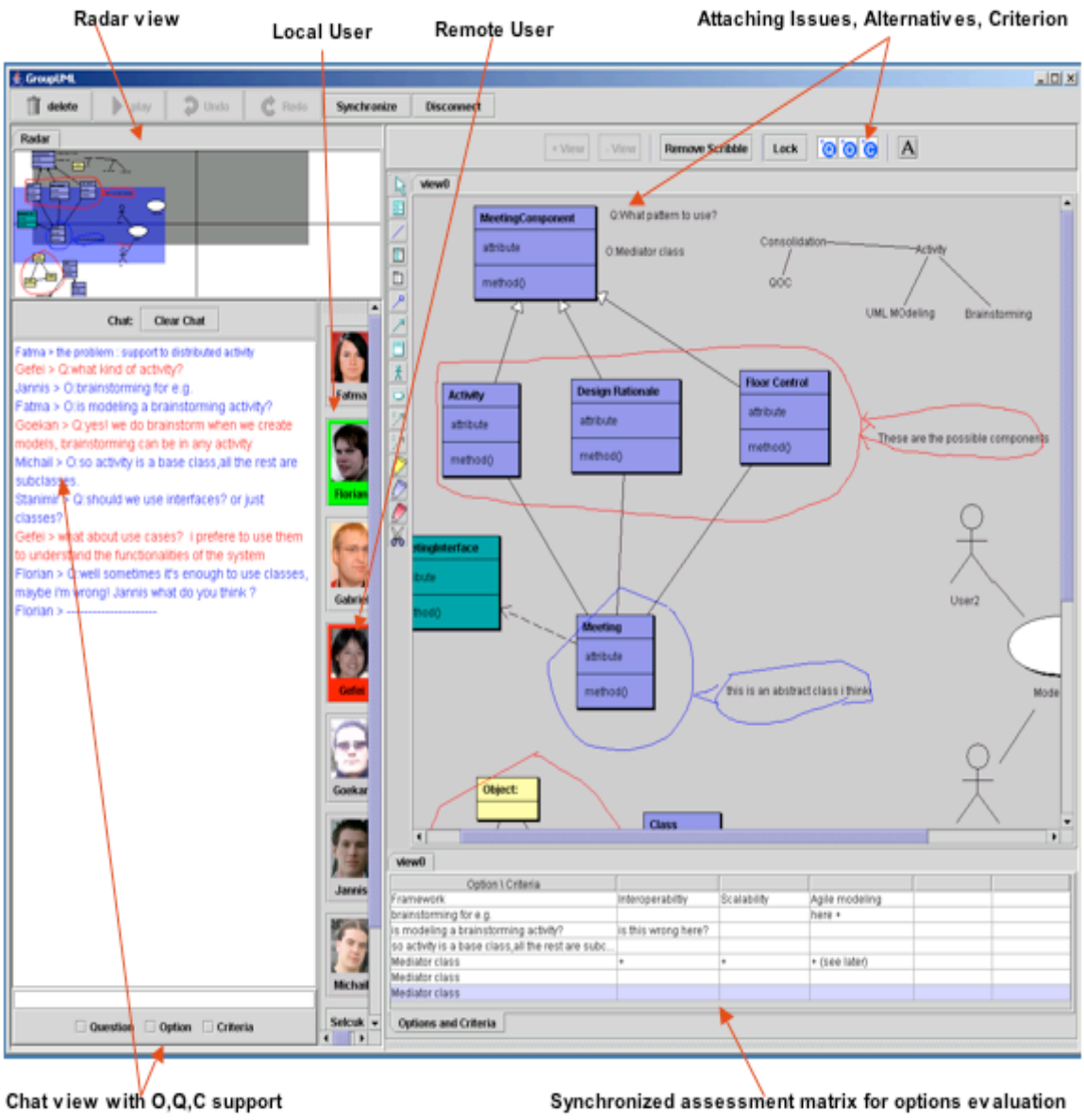


Figure 6.24: GroupUML user interface

total of forty-five senior-level and graduate students with industrial experience background. The emphasis of the case studies were on collaborative issues like group awareness, communication, floor control, and rationale management and their relationships. The subjects were confronted to collaboration issues through the accomplishment of tasks, which lead them into cognitive conflicts and resolutions. The iterative formative evaluation of the case studies was tied to the incremental evolution of the SCOOP prototype, where the results of one iteration of the study were used to modify the prototype before the next study. The final prototype and its design meet the requirements as outlined in chapter 4 (conceptual model)

The SCOOP system is the result of the collaboration of users spread over different locations, which supports the hypothesis of the thesis stating the feasibility of distributed software design and brainstorming according to the vertical process model and requirements engineering in the target development environment.

The case studies have shown that SCOOP framework is a viable solution and validate its ability to support users in carrying out tasks of distributed synchronous software brainstorming and design meetings.

Systems involving remote design, maintenance or diagnosis, such as those including remote expert-worker dialog and communication, are similar to those supporting distributed synchronous meetings. The users of such systems such as blue color workers or road warriors, need to be on the target environment to analyze and understand the problems. They need to experience the system, which is built to support the users carrying their tasks, to discover and identify new requirements and then use these as feedback to improve the system. These systems can use SCOOP's distributed meeting management core component.

We believe that the requirements engineering for such systems, their design, use, and evaluation are interleaved and not as distinct steps in a linear development process that moves from analysis through design to implementation, use and, finally, evaluation.

Consequently, the vision in conducting the case studies in this dissertation is, first, to show that systems based on SCOOP's distributed meeting management core concept can be built and improved using the systems themselves. Second, to show that SCOOP is the mechanism to discover and communicate requirements, and to develop and evaluate alternative solutions for these systems. In particular, distributed synchronous software brainstorming and design needs to be conducted this way.

CHAPTER 7

The BID Approach

We started this research with ill-defined requirements for distributed software design meetings. We used a formative and evolutionary approach where we developed a prototype for distributed real-time software brainstorming and design meetings. Once we had a stable and usable prototype, we shifted our focus on how to evolve the prototype using the prototype itself. New features were designed during a set of case studies organized as a set of distributed meetings.

The lack of requirements prevented us from following approaches like Feature-Driven Design, SCRUM, or RUP. These approaches assume that requirements have already been identified. In our case, we followed an empirical approach to identify the requirements.

During the process of building the prototypes and the underlying SCOOP framework, we also developed a method to understand the requirements identification process, as an incremental self-improving process, and the interaction of the users with the evolving system. This method enabled us to identify the requirements and the design of a flexible architecture to support global brainstorming meetings. We call this method the Bootstrapping Incremental Design

(BID).

7.1 Bootstrapping: Definitions and Concept

Bootstrapping is a general term, that describes any operation which allows a system to generate itself from a small well-defined subset [125].

The term bootstrapping has several connotations in computer science, ranging from binary loaders to compiler construction. Bootstrap most commonly refers to a program that begins the initialization of an operating system, such as GRUB, Lilo or ntldr. A small amount of code is required to start the computer, and it progressively loads more complex code, until the full operating system is available.

In the context of compilers, it is common to define one language as a subset of another, so that subset (S1) is contained in subset (S2) which in turn is contained in subset (S3) and so on, that is:

$$\text{Subset (S1)} \subseteq \text{Subset (S2)} \subseteq \text{Subset (S3)}$$

One can first write a compiler (C1) translating a subset (S1) of Language (L) in machine code (M). Then write a compiler (C2) translating a subset (S2) of (L) using the language subset (S1) of (L), and so on (see Figure 7.1). This defines the compiler bootstrapping process by which a simple compiler for a subset of a programming language is used to translate a more powerful compiler for a larger subset of the language, which in turn may handle an even more complex compiler and so on.

Many compilers for popular languages were first written in another implementation language, and then rewritten in their own source language using this process.

Civil engineers use this concept as well. For instance the highest suspension bridge in the world “Le viaduc de Millau” in France was recently built

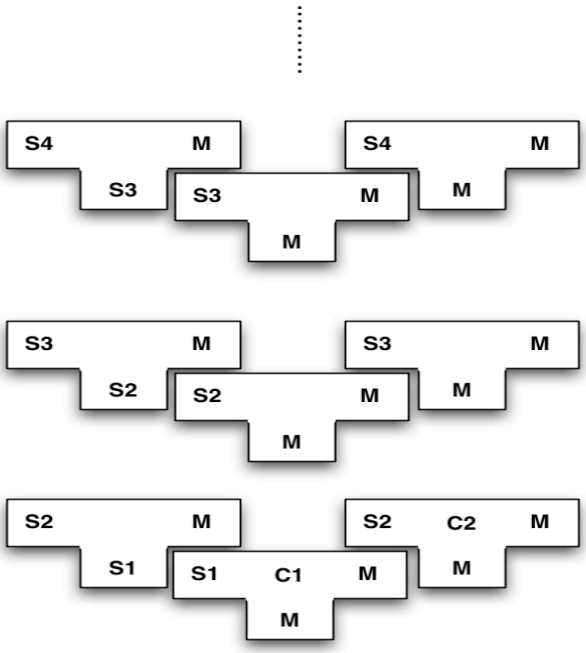


Figure 7.1: Compiler bootstrapping: C1 is the initial handcrafted compiler for a subset S1 of a language L on a target machine M. C1 is used for code generation for S1 into M. C2 is the compiler for S2 written in S1 for the target machine M. We perform this process till getting a compiler for the full subsets of the language L on the target machine L.

using the concept of bootstrapping. Engineers built first the pillars of the bridge, then draw a thin cable across the both hills, then they used that cable to elevate a larger one. Eventually, cables were used to pull a third bigger one and eventually to join both hills. On top of hills, engineers put another layer of pillars that are used to join the wires.

Figure 7.2 shows the application of the bootstrapping process to the development of the SCOOP framework. SCOOP1 (by analogy to the compiler for subset1) is the initial version of SCOOP and GroupUML0 is its initial realization (by analogy the machine language). The result of the use of GroupUML0 are new design and requirements (by analogy to the machine code for the target

machine)

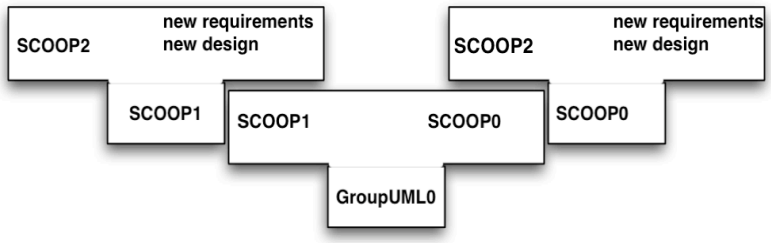


Figure 7.2: bootstrapping SCOOP: GroupUML0 is a handcrafted version of SCOOP1. It was used to develop the next version SCOOP2, which in turn was used to develop the subsequent version. The bootstrapping process continue further until getting a stable version of SCOOP.

Figure 7.3 depicts a UML activity state diagram to describe a general bootstrapping abstract method.

In the next section we introduce and discuss the target community for the SCOOP framework and the use of the bootstrapping approach for designing distributed meeting frameworks.

7.2 Users of distributed software design support systems

Most research systems are built and used by the people who need them. In this section we describe the relevance of our research on developers for distributed systems.

Open-source software is usually created by geographically dispersed developers. Most of these volunteer developers are also the users of the software they create. Open-source programmers rarely meet in person [95]. They rely heavily on electronic media such as electronic mail, the world-wide web and software configuration management systems such as CVS [25]. Open-

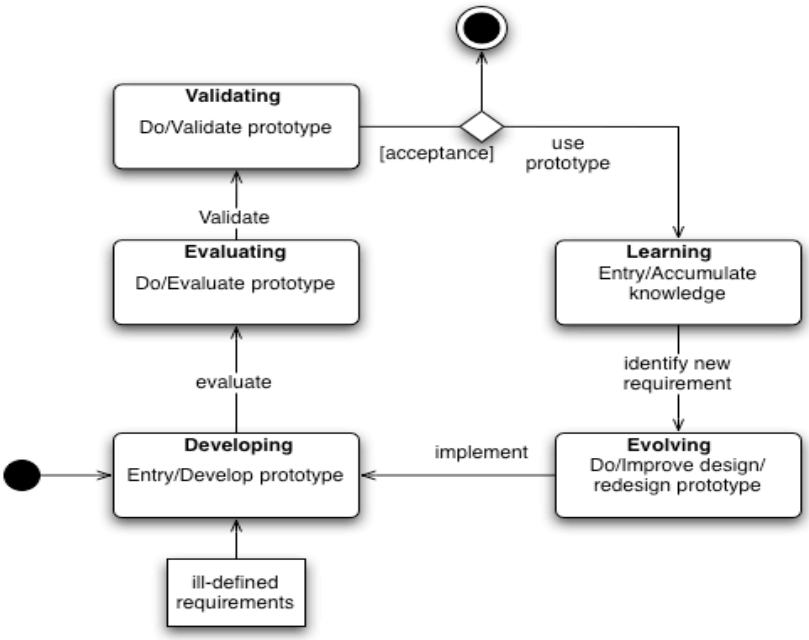


Figure 7.3: Bootstrap abstract method (UML activity state diagram): initially, the requirements are ill-defined but as we progress in the development cycles, we gain better understanding of issues and requirements.

source communities lack synchronous collaboration support and have little real-time knowledge about each other. According to Yamauchi, face-to-face meetings within the open-source community would lead to a drastic improvement to the actual software development process. We believe that providing groupware support for synchronous collaboration to the open-source community helps to minimize several collaborative issues, such as duplicate work and misunderstanding, and to enforce achieving agreement and building consensus among developers. Hence, among primary target users of the SCOOP framework could be the open-source community benefiting from synchronous collaborative software design meetings.

Researchers in the field of distributed software development can benefit from SCOOP as well to conduct activities that are based on distributed meetings management. SCOOP may be also used in designing and conducting controlled experiments based on distributed synchronous meetings.

Researchers of Model-Driven Development (MDD) may use SCOOP and the BID approach to design new functionalities by building models, compile the models at design time and integrate the executable into the system that generate them at run-time. That is, without shutting down the used system, one can design a functionality, automatically compile it and the executable integrated into the system whilst running.

Researchers investigating distributed applications such as DWARF [85] may also use SCOOP and BID to design mobile augmented reality applications. In particular, augmented reality systems for mobile remote maintenance could benefit from the BID approach [108] to design and improve tools used for similar tasks. This is because mobile remote maintenance applications are similar to distributed collaborative meetings, which are supported by SCOOP.

7.3 The Bootstrapping Incremental Development Approach

In this section we describe steps of the BID empirical approach as shown in Figure 7.3. A step in the BID approach (BID-step) is a case study as described in chapter 6 (case studies, see figure 6.5).

Step 0: bootstrap start

Step 0 is the start point of the bootstrapping process. We use the scenario described in chapter 3 and the user-centered-design approach to design and build the initial prototype system [98]. The initial prototype had basic capabilities enabling synchronous shared scribbling and text writing using smart boards [99]. Different meeting rooms accessing the same server could manipulate the

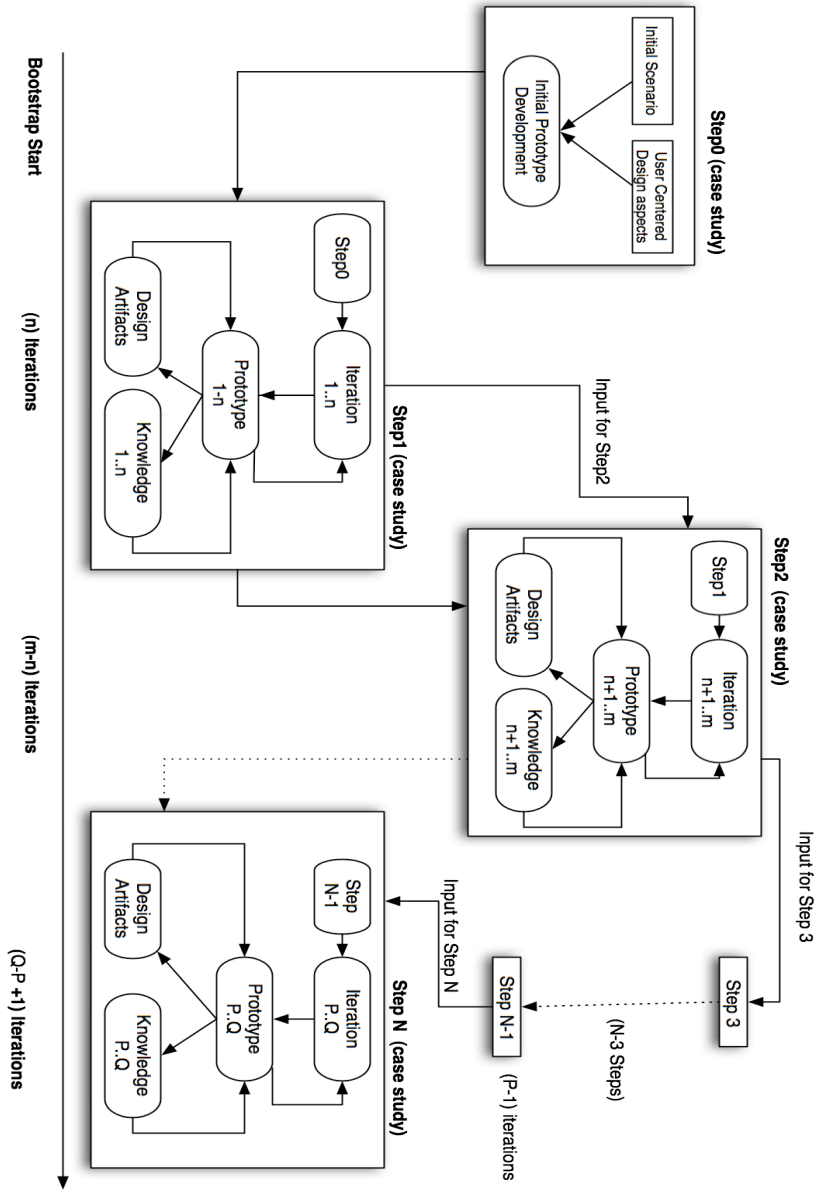


Figure 7.4: Bootstrapping Incremental Design Process (BID): BID is a bootstrapping, a sequence of BID Steps, each of them is also a bootstrap process called inner bootstrap process.

same artifacts by drawing and clicking on the smart board. A keyboard enabled users to enter text elements associated with the artifacts. Changes were immediately propagated to the other sites.

Step 1: first step in the bootstrapping process

Based on the knowledge gained from Step0, we conduct a Step1 meeting. A Step is a set of case study that consists of n iterations, that is, n meetings.

Each of these iterations is a distributed meeting conducted to brainstorm and design the requirements of the prototype itself with the goal of getting the desired state of the design, an improved design artifact, and an improved prototype [104].

The prototype evolves from one meeting to another, eventually having n times improved a given functionality or developing new ones. Prototype n denotes the n-th prototype.

Design artifacts are the revised SCOOP models designed by the participants utilizing the prototype to improve the behavior of the prototype. The models describe functionalities that the participants need at that moment but are not yet available. For example, a design artifact can be the awareness model designed during an iteration, incorporated into the system prototype, used in a subsequent iterations. In n iterations, a design artifact can be changing n times. Design artifacts may exist only temporarily and disappear as soon as the functionality is built and is incorporated into the system. However, its creation and evolution history is kept as another design artifact in the system.

With knowledge we denote the improved understanding of the requirements of the system. Knowledge can be accumulated over a set of meetings. Knowledge is used in redesigning the prototype in subsequent BID-steps, that is, subsequent case studies. Typically, knowledge n in Step 1, which is produced after n iterations, will serve as the basis for a BID-step 2 and will evolve to

Knowledge m after (m-n) iterations.

Step N: Nth step in the bootstrapping process

BID-step N takes place based on the results of the previous BID-step N-1, which takes p-1 iterations to complete (see Figure 7.5).

Using the prototype and knowledge from Step N-1, one or several design artifacts are produced and are fed back to the prototype. The knowledge produced during iterations (p..q) is used to redesign the prototype utilizing the design artifacts. At this level, we iterate over Step N (q-p+1) times until we have the desired state of the prototype. A desired state is an accepted functionality of the system.

Although the design process is not predictable, the knowledge p necessarily remains the same in the worst case and evolves to knowledge q in the best case. Knowledge is not quantifiable. But still, we can prove the assumption that in general:

$$\text{knowledge}_{(q)}(\text{Step}(N)) >= \text{knowledge}_{(p)}(\text{Step}(N-1)).$$

This is a straightforward claim, since we need more knowledge to move from current Step N-1 to the Step N; if we don't produce more knowledge, we are stalled in step N-1.

It is interesting to notice that each Step of the BID approach is a bootstrapping process in itself. To arrive a final system, we conduct several meetings in which we incrementally iterate using an inner bootstrapping process, hence the name Bootstrapping Incremental Design process.

The bootstrapping process is shown in figure 7.5. A UML meta-model description of the bootstrap process is shown in figure 7.6. A bootstrapping process is basically a set of activities that are build upon each other starting from a basic state. *Knowledge acquisition* and *system improvement* are the characteristics of a bootstrapping step. The *System improvement* activity yields an improved prototype for the subsequent bootstrapping step using knowledge acquired dur-

ing the current bootstrapping step.

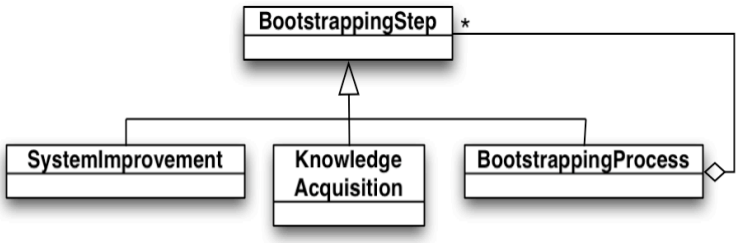


Figure 7.5: A Bootstrapping Meta-Model

Each of the case studies described in chapter 6 were based on a complete step of the BID process described above. It consisted of several development meetings. An elaboration of the iteration Step N in the inner bootstrapping process is presented in figure 7.5.

7.3.1 Related Work to the BID approach

We will now argue that the BID approach can be integrated into other methodologies such as Scrum, RUP, and FDD.

The term "Scrum" denotes a teamwork strategy in the game of rugby with the goal of "getting an out-of play ball back into the game" [138].

It was first proposed for managing industrial processes by Takeuchi and Nonaka who presented an adaptive, quick, self-organizing product development process [138]. It is an empirical approach applying the ideas of industrial process control theory to systems development that introduces the ideas of flexibility, adaptability and productivity. It does not define any specific software development techniques for the implementation phase. Scrum concentrates on how the team members should function in order to produce the system flexibly in a constantly changing environment.

The main idea of Scrum is that systems development involves several

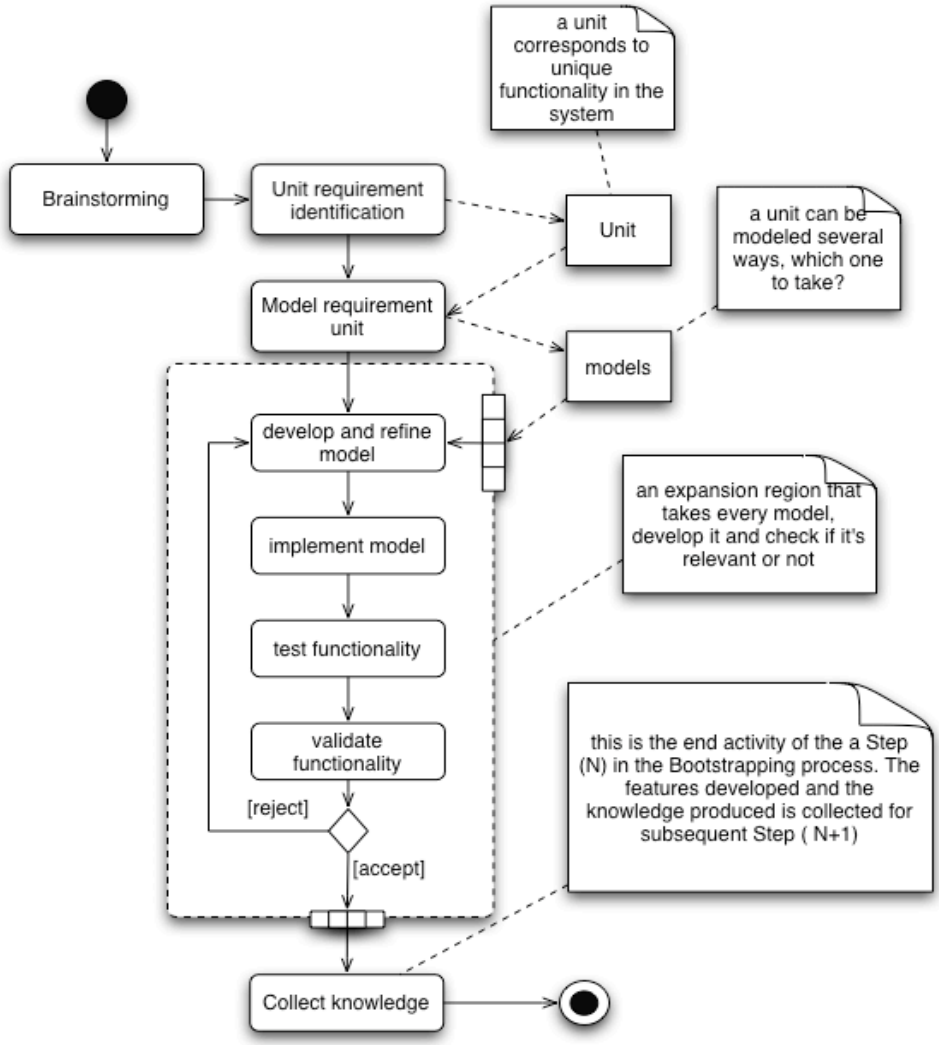


Figure 7.6: An elaborated model of the inner bootstrapping process of the BID approach (UML activity diagram).

environmental and technical variables (e.g. requirements, available time frame, resources, and technology) that are likely to change during the process. This makes the development process unpredictable and complex, requiring flexibility

of the systems development process to be able to respond to the changes.

Scrum has been used to improve existing engineering processes such as testing practices in an organization, because it involves management activities aiming at consistently identifying any deficiencies or impediments in the development process as well as the practices that are used. Scrum process includes three phases: pre-game, development and post-game.

During the pre-game phase, the planning and architecture take place. Planning includes the definition of all open issues for the system being developed. this is called the product backlog list is created containing all the requirements that are currently know.

The Scrum user scrutinizes the backlog list, which is considered as a container for the requirements, based on the current items in the list, plan and build a high level design of the system. In the BID approach, the requirements are identified according to the current step and are further identified and developed during subsequent steps. Therefore, BID can be used to discover and then enrich the backlog list with new requirements when designing the system.

During development phase, the system is developed in Sprints. Sprints are incremental cycles where the functionality is developed or enhanced to produce new increments.

The post-game phase contains the closure of the release. This phase is entered when an agreement has been made that the environmental variables such as the requirements are completed. In this case, no more items and issues can be found nor can any new ones be invented. The system is now ready for the release and the preparation for this is done during the post-game phase, including the tasks such as the integration, system testing and documentation

In the BID approach, the incremental identification of the requirement and the development of the system happens at the same time. Moreover, requirements helps to improve the development of the current system and the system is used to identify or improve the requirements process. Hence, BID can be integrated into Scrum to support the design and improvement of systems at run-

time also.

The Rational Unified Process (RUP) was developed by Phillipe Kruchten, Ivar Jacobson and others at Rational Corporation to complement the UML, notation with an industry-standard software modeling process [77].

RUP is an iterative approach for developing object-oriented systems, and it strongly embraces use cases for modeling requirements and building the foundation of the system.

A typical project is divided into four phases called Inception, Elaboration, Construction and Transition (see Figure 7.5). These phases are split into iterations, each having the purpose of producing a demonstrable piece of software. The duration of an iteration may vary from two weeks or up to six months.

The BID approach can be used in the elaboration and construction phases of RUP. During the elaboration phase, RUP assumes that requirements are stable enough to build a solid architecture and a solid software project plan. After this phase, most use cases and all actors should have been identified and described, the software architecture is described, and an executable prototype of the architecture is created.

In the construction phase, all remaining components and application features are developed and integrated into the product, and tested. RUP considers the construction phase a manufacturing process. By making the developed prototypes available to the construction phase, BID can be used as an extension to RUP. BID can improve the elaboration phase by helping in identifying and developing the requirements and preparing to move the construction phase.

Feature Driven Development (FDD) is another agile approach for developing systems. However, unlike the RUP, the FDD approach does not cover the entire software development process, but rather focuses on the design and build phases [115]. FDD was first reported by Peter Coad and Jeff Luca in [24] and further developed by, Peter Coad and Stephen Palmer [115].

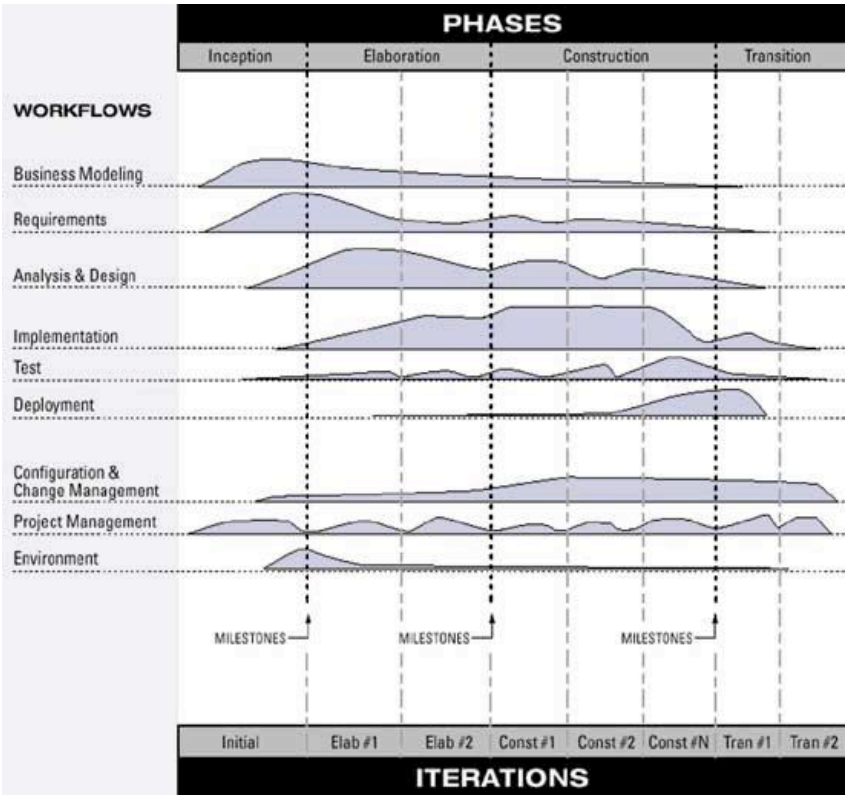


Figure 7.7: RUP Process

FDD consists of five sequential processes during which the design and build of the system is carried out (see Figure 7.7) by developing an overall object model, and a feature list. From this list a small group of features is selected and feature teams are formed to develop the selected features. The design and build by feature processes are conducted iteratively, during which the selected features are produced.

BID and FDD share the focus on the design and building processes. FDD is built upon a well defined set of activities, whereas BID influences the activi-

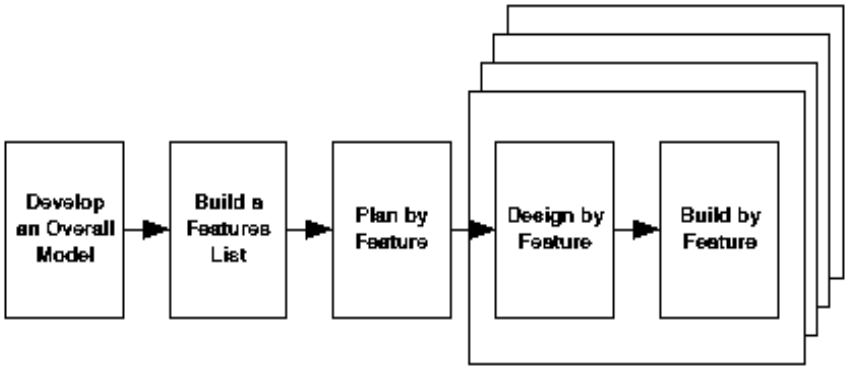


Figure 7.8: Process of FDD

ties by the current state of the requirements, the prototypes, and the models created. FDD is based on a features list that is created after walkthrough of each domain areas. The feature list serves to create high-level plans in which the feature sets are sequenced according to their priority and dependencies. BID can support FDD in improving the understanding of the requirements by making them explicit through prototyping.

7.3.2 Applying the BID Approach

Up to this point, we presented an approach for software design and development where requirements are ill-defined. A large part of the BID approach is based on experimental processes, where users are part of the development process.

We now go one step further into making the BID approach explicit by applying it to the design and development of the SCOOP framework.

For the initial Step we used the user-centered design approach to design the handcrafted version of SCOOP. For the rest of the bootstrapping BID-Steps we

used a participative design approach, where users use the current version of SCOOP to design next version of SCOOP.

Step 0: bootstrap start

To trigger the bootstrapping process, we conducted an initial distributed

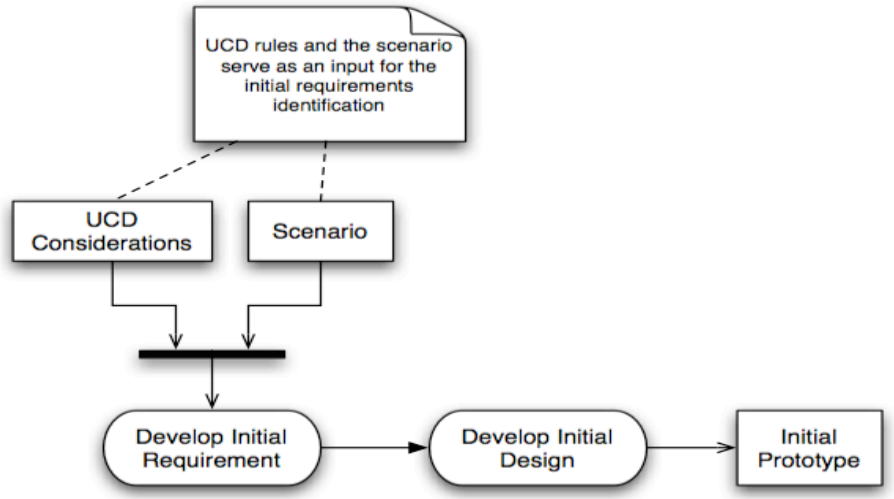


Figure 7.9: bootstrapping initial step Step0: using user-centered design approach

session of problem solving and gathering requirements through interviews using the following questions:

- What sorts of tasks must be supported and mediated by the initial prototype?
- What types of infrastructure and technology are acceptable for use to users in carrying out their tasks?

The first question was answered through the scenario described in chapter 4. From the scenario we derived the core requirements such as support for sharing artifacts among distributed developers, supporting communication, and enabling awareness information.

To answer the second question, we used hardware such as electronic white boards (smart boards) for input and networked computers for distributed communication. The smart boards in particular were helpful to conduct design and brainstorming meetings because users cold user them like traditional white boards.

As mentioned earlier, User-Centered Design approach (UCD) was used in the an development of the initial as shown in Figure 7.10.

UCD is an approach that places the user at the center. UCD focuses on the artifacts being designed (e.g., the object, communication, space, interface, service, etc.). Looking for ways to ensure that it meets the needs of the user. UCD seeks to answer questions about users and their tasks and goals, then use the findings, that is, the answers, feedback, and the initial exploration, to drive development and design. UCD seeks to answer questions such as:

- Who are the users of this system?
- What are the users' tasks and goals?
- What are the users' experience levels with this system?
- What functions do the users need from this system?
- What information might the users need, and in what form do they need it?
- How do users think this system should work?
- How can the design of this system facilitate users' cognitive processes?

UCD can be used to drive the improvement of the usability and usefulness of everything from "everyday things" [35] to the usefulness of software. Usefulness of a system relates to its relevance to the user, whereas usability of a systems relates to its ease-of-use. A researcher collects primary data or uses secondary sources to learn about the needs of the user. This information is often interpreted in the form of design criteria or requirements. The designer interprets these criteria, typically through concept sketches or scenarios. The focus moves

on to the development of the solution.

In user-centered design, the roles of the researcher and the designer are distinct, but still interdependent. The user is usually not part of the team, but is represented by the researcher. To test a specific design, the team puts together a preliminary version which can be as simple as pieces of paper with proposed design sketches, or designs that look like finished products

Step N: Nth step in the bootstrapping process

The initial step Step0 described above triggered the bootstrapping process of the design of SCOOP. Now we describe Step N of the bootstrapping approach.

The main issue is how to use the prototype from Step N-1, how to develop the increments, how to contribute to design, and to evolve of the whole system.

The initial prototype of SCOOP served as a platform to conduct distributed software design meetings where the design challenge was the design of SCOOP itself.

For the bootstrapping step N we are benefiting from the participative design approach. Since the users of the system can express and can model their requirements in a better way than designers [3], involving potential and skilled users in the design is important.

The field of participatory design grew out of work beginning in the early 1970s in Norway, when computer professionals worked with members of the Iron and Metalworkers Union to enable the workers to have more influence on the design and introduction of computer systems into the workplace.

In participatory experiments, the roles of the designer and the researcher may overlap and the user becomes a critical component of the process. People express themselves and participate directly and operatively in the development process. The new situation requires adaptation. In particular, there is a need to new tools and new methods taking into consideration the active user participa-

tion into the process.

Figure 7.11 shows a description of SCOOP design using the BID approach.

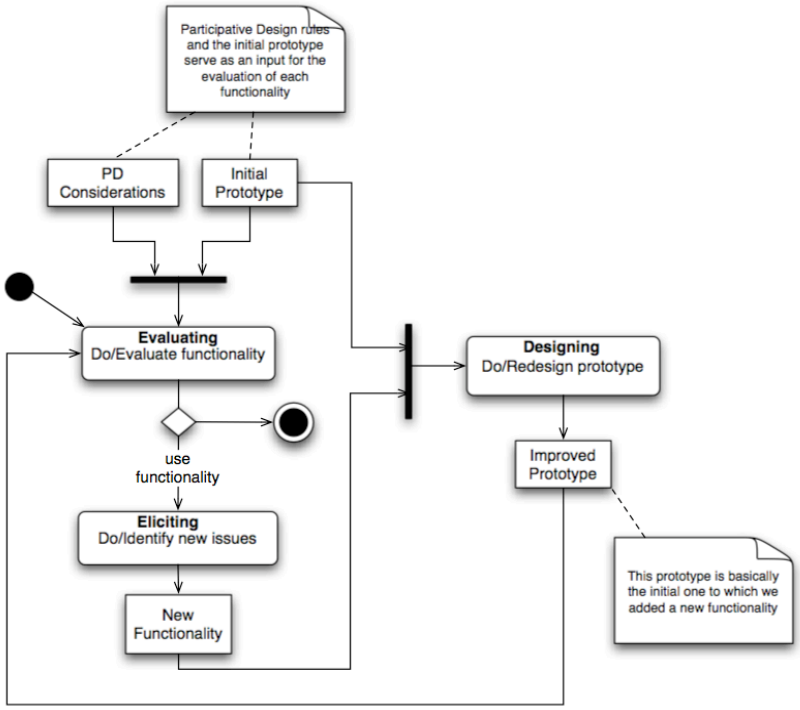


Figure 7.10: Participative design with the bootstrapping approach

The user-centered design process corresponds to the initial bootstrapping step that triggers the bootstrapping process. The participative design process is a design process "augmented" with user participation. The iterative and participative nature of the bootstrapping process is an improvement of the traditional participative process to a bootstrapping participative process.

7.3.3 Conclusion for the BID approach

We presented in this chapter the BID (bootstrapping Incremental Design)

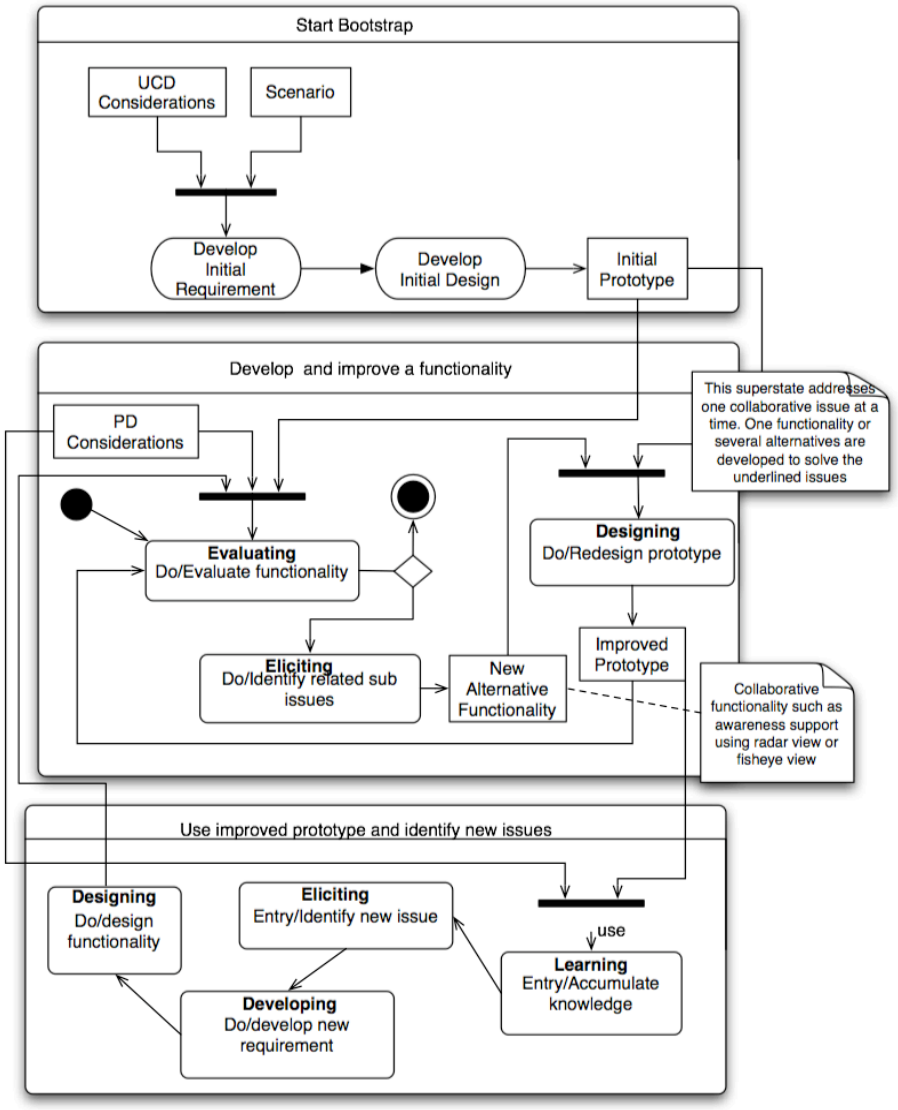


Figure 7.11: Applying the bootstrapping design approach in designing the SCOOP framework

approach, an approach applying the idea of bootstrapping theory to systems

development.

The BID method provides explicit guidance on eliciting the architectural requirements, designing the architecture, and analyzing the resulting design. Moreover, it provides software architects with a framework for understanding the technical trade-off and risks they face as they make architectural design decisions.

CHAPTER 8

Conclusion and Outlook

This thesis explored the requirements for a framework for global and distributed software development, and in particular for distributed collaborative software brainstorming and design meetings in same time/ different place settings. Our hypothesis was that distributed software brainstorming and design meetings can be conducted according to the collaborative vertical process model defined in chapter 3.

To show the feasibility of distributed modeling meetings, we designed and conducted several case studies. This led to the identification of several collaborative work issues, from which we derived requirements for the development of GroupUML and the SCOOP framework. GroupUML is a groupware application that enables distributed participants to conduct distributed synchronous software brainstorming and design meetings. It supports collaborative work aspects such as group awareness, group communication, group activity, group memory, floor control, location transparency, and rationale knowledge management.

These collaborative work aspects, issues, and requirements have laid the

ground for the following achievements:

We developed -through a scenario-driven requirements elicitation method- an initial model describing the collaborative activities of distributed software brainstorming and design meetings. The break down of these activities into sub-activities such as Initial Model Creation, Model Transformation, Conflict Identification and Resolution, and Consolidation is described in chapter 4. We also developed a SCOOP object model based on the Model View Control (MVC) pattern to ensure the separation of concern: Each of the identified collaborative work aspects corresponds a set of model objects, which encapsulates the application data, and view objects for the corresponding visual representation.

We designed SCOOP as a collection of reusable component objects representing Activity, Location, FloorControl, Rationale, GroupAwareness, Communication, and GroupMemory, each with well defined responsibilities described via contracts. Contract-based style components development is a useful formalism that expresses high-level specification of the components' behavior. Different implementations can be provided that show different behavior (for example, different locking mechanisms within the same application) or adapting a component to fulfill a special requirement in an application having the same domain as SCOOP.

We conducted a series of exploratory case studies following a formative empirical method not only as an evaluation approach but also as a process of exploring, incrementally designing, and evaluating new functionality of SCOOP.

Starting with deployment of the initial SCOOP prototype in the target environment, namely the distributed sites, the participants used the prototype to design new activities and new functionalities for another version of SCOOP. This way, the participants used the initial SCOOP prototype to improve SCOOP itself in an iterative way resulting in the incremental evolution of SCOOP.

At the end of the dissertation we described a process resulting from the case studies we conducted throughout this research and from the development of SCOOP framework.

We used a formative and evolutionary approach to develop the prototypes used during cases studies in an attempt to identify the requirements for distributed real-time software brainstorming and design meetings. The current state of each prototype and the knowledge gained from the case studies influenced the subsequent design meetings. The new features were designed and added during the case studies. The improvements were made incrementally while conducting distributed meetings for designing the missing features.

During the process of building the framework, we were made aware of the possibility in shaping the method we were using to understand the requirements identification process. We identified an incremental approach with three key ingredients: user participation, evolutionary experimentation, and a bootstrapping-like design method. This mix enabled the identification of the requirements and the design of a flexible architecture to support distributed brainstorming meetings. We call this method the Bootstrapping Incremental Design (BID).

Future directions

The models, the case studies, and the SCOOP framework described in this dissertation, can be developed in several directions. First, SCOOP can be evaluated in other application domains such as remote maintenance, diagnosis, or expert systems, where the users are mobile and need the expertise of remote experts. The requirements for such systems place emphasis on knowledge exchange; differ from software design meetings, where the emphasis is on software design activity, yet both share the distributed meeting management as a crucial component. As such, SCOOP could be investigated for its feasibility as a

support tool for remote meetings with experts.

The integration of SCOOP and applications such as augmented reality-based maintenance systems, involving remote expertise, data exchange, and distributed coordination is another possible area for investigation.

Another possibility is the integration of BID with the Unified Process, and agile methods such as SCRUM, to assess the impact of the use of BID in distributed agile-based development projects.

The extension of the case studies to population different from software developers, for example, to blue collar workers is another challenging problem. Blue collar workers have different practices, motivations, and interests than developers. This can be investigated in a study of SCOOP's ability to handle acceptance issues not only of technical nature. Social issues, working habits, tight schedules, and peoples' reluctance and inflexibility may question the acceptance of the SCOOP framework where collaboration and learning within the group is mandatory.

Finally, it might be interesting to study the implications of the deployment of SCOOP applications for privacy and security issues. Conducting geographically distributed meetings may raise privacy issues, because users may be unwilling to participate actively in brainstorming and sharing ideas, their ideas are recorded and made available for unauthorized users. Even within the distributed team, they might be reluctant to use SCOOP because of confidential meetings, sensitive business data, and non-disclosure agreements. This might also have an impact on the acceptance of the SCOOP framework.

Appendix A

Research Material

Issues categories	Group A: number of users mentioned issues out of total users.	Group B: number of users mentioned issues out of total users.	Proposed Solutions	Retained Solutions
Communication	6/7	5/7	3	2
Awareness	1/7	1/7	0	0
Floor control	1/7	0/7	1	1
Rationale Management	0	0	0	0
Collaboration with peers (as a whole)	2/7	3/7		
Satisfaction with GroupUML use	4/7	3/7	4	3

A .1: Analysis of the answers of participants to the Questionnaire I: the categories reflect the requirements for collaborative software design and brainstorming identified in chapter 4

Iterations	Communication issues (Summary)
Analysis of the participants answers and suggestions to the questionnaires	•Chat possibility •Integrating communication within diagram models •Communication mean should not be tied to a particular view but to all workspace •Chat window integrated into workspace and not as standalone •Possibility to hide and show communication window •Voice/audio channel
Implemented Functionality	•Chat/instant messaging window available to all views •Connect temporary messages to diagram models •Hide & show chat window
Direct resulting issues of the incorporation of functionalities	•Crowded text in chat window •Cannot distinguish sent and received messages of local/remote participants •No private messages among participants
Further Improvement steps	•Added colors to distinguish remote and local group messages. •Added context of message: users, relative view, model, and possibly current object under update. •Added audio conference channel. •Private messages are not relevant in synchronous collaboration.

A .2: Summary of the qualitative data about communication issues, the corresponding solutions, successive resulting issues, further improvements, and the final result and impact on the collaboration process

Final results / Impact on collaboration process	From the analysis of the videotaped sessions and the answers to questionnaires, it is noticed that *chat* is used most of the time along the meetings. Audio channel is only used when dealing with difficult issues Positive impact of the current *communication issues iteration* on collaborative software brainstorming and design: • Integration of chat and communication within the design activity: • Improved and enabled a fluid collaboration. • Lead to a stable communication mean in GroupUML has been identified • Provided awareness information (indirect side effect)

A .2: Summary of the qualitative data about communication issues, the corresponding solutions, successive resulting issues, further improvements, and the final result and impact on the collaboration process

Phases	Floor Control issues (Summary)
Analysis of the participants answers and suggestions to the questionnaires	•Introduce a synchronization mode •Introduce role or moderator to manage turn taking •Introduce an explicit lock/unlock functionality •Introduce a timer when initiating an update of an artifact
Implemented Functionality	• Lock/unlock functionality (several levels: artifact, group of artifacts, a view, or workspace)
Direct resulting issues of the incorporation of functionalities	•Artifacts are locked until the initiator unlock them. This cause other participants to wait till artifact are released • Users forget to unlock when finishing an update • Lock the whole workspace caused the interruption of the work process
Further Improvement steps	•Added a timer to the lock functionality so the initiator does not keep the floor as more than as necessary •Added blinking behavior to the lock to remind users of their locking •Fine grained lock •Name of the current user having the current floor is displayed to all users
Final results / Impact on collaboration process	•From the analysis of the videotaped sessions and the answers to questionnaires, it is noticed that the *floor control* policies were well welcomed by participants. •Positive impact of the current *floor control policies* on collaborative software brainstorming and design: •Provided support to coordinate users actions and to avoid waste of time and frustration of the users •Provided awareness information (indirect side effect)

A .3: Summary of the qualitative data about floor control issues, the corresponding solutions, successive resulting issues, further improvements, and the final result and impact on the collaboration process

Categories	Group A (number of users mentioned issues out of total users)	Group B (users number mentioned issues out of total users)	Proposed Solutions	Retained Solutions
Communication issues identified	0/7	1/7	0	0
Awareness issues identified	5/7	4/7	5	2
Floor control issues identified	1/7	0/7	1	1
Rationale Management issues identified	2/7	3/7	0	0
Collaboration with peers (as a whole)	4/7	5/7		
Satisfaction with GroupUML use	5/7	6/7	6	3

A .4: Analysis of the answers of participants to the Questionnaire of chapter 6, figure 6.15: we notice more emphasis on awareness issues than on floor control or rationale management issues.

Phases	**Awareness issues (Summary)**
Analysis of the participants answers and suggestions to the questionnaires	•Integrating a list of users names and pictures participating in the collaboration •Highlighting current user picture •Highlight objects being updated •Current activity status (e.g. “syncing”, “editing”, “creating”, and so on.) •Multi-pointers •Bird view
Implemented Functionality	•Integrating a list of users names and pictures participating in the collaboration •Highlighting active user picture •Highlight objects being updated
Direct resulting issues of the incorporation of functionalities	•Users forget sometimes to highlight their picture when starting an action. Therefore, other users actions can be accounted for the users already highlighted.
Further Improvement steps	•Radar view •Current activity status •Highlighted users are reset after a time-out so users have to highlight their •Blinking objects being updated

A .5: Summary of the qualitative data about awareness issues, the corresponding envisaged solutions, their successive resulting issues, the improvement, and the final result and impact on the collaboration process

Final results / Impact on collaboration process	From the analysis of the videotaped sessions and the answers to questionnaires, it is noticed that *awareness information* is the most useful information that enabled the coordination of the design activities of users. Positive impact of the current *communication issues iteration* on collaborative software brainstorming and design: • Integration of awareness information within the design activity: • Improved and enabled a more reliable collaboration.

A .5: Summary of the qualitative data about awareness issues, the corresponding envisaged solutions, their successive resulting issues, the improvement, and the final result and impact on the collaboration process

Phases	Rationale Management issues (Summary)
Analysis of the participants answers and suggestions to the questionnaires	•Integrating rationale to the models themselves •Adding text description to current view of the created models •Forcing users to enter rationale before moving to another activity straight after having created objects
Implemented Functionality	•Integrating rationale management within the design activity •Added text description of reasons for creating artifact when it is not obvious •Forcing user to enter rationale is ignored • Added a gIBIS support
Direct resulting issues of the incorporation of functionalities	• Crowded models •Manual entry of rationale in the assessment matrix is not considered fully as part of design activity
Further Improvement steps	•Connect rationale in the form of issues, options, and arguments to design models •Connected rationale information is parsed automatically and entered to the assessment matrix •Show and hide rationale information helps to avoid a crowded workspace
Final results / Impact on collaboration process	The analysis of the videotaped sessions and the answers to questionnaires, showed that *rationale management* is difficult to manage in the first glance but vital to support short-term and long-term meetings where decisions and argumentation are made explicit and are part of the models created through collaboration. Positive impact of the current *rationale issues iteration* on collaborative software brainstorming and design: • Integration of rationale information management within the design activity and not as an activity apart as rationale management systems used to be. •Improved support to users in documenting their design process and provide necessary information background to built and history of the meetings.

A .6: Summary of the qualitative data about rationale issues, the corresponding envisaged solutions, their successive resulting issues, the improvement, and the final result and impact on the collaboration process

Categories	Group A (users number mentioned issues out of total users)	Group B (number of users mentioned issues out of total users)	Proposed Solutions	Retained Solutions
Communication issues identified	3/7	2/7	0	0
Awareness issues	2/7	1/7	0	0
Floor control issues	0/7	0/7	0	0
Rationale Management issues	7/7	6/7	8	3
Collaboration with peers (as a whole)	6/7	5/7		
Satisfaction with GroupUML use	5/7	6/7	8	3

A .7: Analysis of the answers of participants to the Questionnaire of figure 6.6 chapter 6.

Appendix B

Abbreviations

AND Alcatel Data Network

BID Bootstrapping Incremental Design

COM Component Object Model

CORBA Common Object Request Broker

CSCW Computer Supported Cooperative Work

CVS Concurrent Version System

DWARF Distributed Wearable Augmented Reality Framework

EMS Electronic Meetings Systems

FDD Feature-Driven Design

FR Fault Report

gIBIS Graphical representation of IBIS

GSD Global Software Development

GSE Global Software Engineering

HCI Human Computer Interaction

IBIS Issue-Based Information System

ICICLE Intelligent Code Inspection Environment in a C Language Environment

LAN Local Area Network

MDA Model-Driven Systems

MDD Model-Driven Development

MOF Meta Object Facility

MVC Model View Controller

OMG Object Management Group

OOActSM Object-Oriented Activity Support

QOC Questions, Options and Criteria

RAD Rationale Analysis Document

RFC Request For Change

rIBIS Real-time IBIS

RMI Remote Method Invocation

ROI Return On Investment

RPC Remote Procedure Call

RUP Rationale Unified Process

SCOOP Synchronous Object-Oriented Process

SDD System Design Document

SOAP Simple Object Access Protocol

UML Unified Modeling Language

WAN Wide Area Network

WYSIWIS What You See Is What I See

WYSIWITYS What You See Is What I Think You See

XMI XML Metadata Interchange

XML extensible Markup Language

XP eXtreme Programming

Bibliography

[1] A.Newell & H. Simon, The Theory of Human Problem Solving; reprinted in Collins & Smith (eds.), Readings in Cognitive Science, section 1.3.

[2] Adams, Paul (2004) A Collaboration Environment to Support Distributed eXtreme Programming. In: FACS PhD Poster Workshop, 11 Nov 2004, Lincoln.

[3] Andrew Clement, Peter Van den Besselaar: Proceedings of the Eighth Conference on Participatory Design: Artful Integration: Interweaving Media, Materials and Practices, PDC 2004, Toronto, Ontario, Canada, July 27-31, 2004

[4] Aoyama, M. Agile Software Process Model. In Proceedings of the 21st IEEE International Computer Software and Applications Conference COMPSAC '97 (1997), 454-459.

[5] Archer, L.B., "Systematic method for designers," in Developments in Design Methodology, Cross, N., Ed. John Wiley & Sons, 1984, chap. 1.3, pp. 57-82, Originally published by The Design Council, London (1965).

[6] B. Bruegge, A.H. Dutoit, R. Kobylinski, G. Teubner.Transatlantic Project Courses in a University Environment 7th Asia-Pacific Software Engineering Conference (APSEC 2000), Singapore, December, 2000.

[7] B.Brügge, A. Dutoit Distributed Software Engineering: Research Issues & State-of-the-art.

[8] B.G. Glaser and A.L. Strauss, The Discovery of Grounded Theory: Strategies for Qualitative Research. Aldine Publishing, 1967.

[9] Beck, K. and R. Johnson (1994). Patterns Generate Architecture. European Conference on Object Oriented Programming. Berlin: Springer-Verlag.

[10] Bellotti, V., & MacLean, A. (1994). Integrating and Communicating Design Perspectives with QOC Design Rationale. Esprit Basic Research Action 7040, AMODEUS-2, Working Paper, ID/WP29.

[11] Bertrand Meyer in "A Conversation with Bartering Meyer, Part II by Bill Veneers December 8, 2003".

[12] Bjørn Erik Munkvold, Robert Anson. Organizational Adoption and Diffusion of Electronic Meeting Systems: A Case Study.

[13] BNF and EBNF : What are they and how do they work? http://www.garshol.priv.no/download/text/bnf.html (June 2005).

[14] Borghoff, U. M. and Schlichter, J. H. (2000). Computer-Supported Cooperative Work: Introduction to Distributed Applications.

[15] Bradley, G, "Control vs. Creativity: Software Engineering at a Crossroads," in Human Aspects in Computing: Design and Use of Interactive Systems and Work with Terminals, Bullinger, H.J., 1991, pp. 561-565.

[16] Bran Selic. The Pragmatics of Model-Driven Development IBM Rational Software http://doi.ieeecomputersociety.org/10.1109/MS.2003.1231145 . Copyright 2004 IEEE Inc.

[17] Brooks, F. P. (1995). The Mythical Man-month. Addison Wesley, San Diego, 2nd. edition.

[18] Brothers, L., Sembugamoorthy, V., Muller, M., "ICICLE: Groupware for Code Inspection." CSCW 90: Proceedings of the Conference on Computer-Supported Cooperative Work, Los Angeles, CA: ACM, 1990, pp. 169-181.

[19] Carolyn B. Seaman, Qualitative Methods in Empirical Studies of Software Engineering. IEEE transactions on software engineering, vol. 25, no. 4, july/august 1999.

[20] Chacon-Moscoso, S., M. T. Anguera-Argilaga, J. Antonio, P. Gil and F.P. Holgado-Tello (2002), 'A mutual catalytic role of formative evaluation: the interdependent roles of evaluators and local programme practitioners', Evaluation 8(4): 413-432.

[21] Christof Ebert, Philip De Neve Surviving Global Software Development. IEEE Software Magazine 2001-03 pp 62-69

[22] Clement, A. and den Besselaur, P.V., "A Retrospective Look at PD Projects," Communications of the ACM, vol. 36, no. 4, 29-37, Jun. 1993.

[23] Clever. Guareis de Farias, L. Ferreira Pires, M. van Sinderen. A conceptual model for the development of CSCW systems. 4th International Conference

on the Design of Cooperative Systems (COOP 2000). Sophia Antipolis, France, May 2000.

[24] Coad, P.,LeFebvre, E. and De Luca, J. (2000). Java Modeling In Color With UML: Enterprise Components and Process. Prentice Hall.

[25] Concurrent Version System http://www.gnu.org/software/cvs/ (june 2005)

[26] Conference Report, Building for People, 1965 UK Ministry of Public Building and Works, London, 1965.

[27] Cramton, C. D. (1997). Information Problems in Dispersed Teams. Paper presented at the Annual Meeting of the Academy of Management (Best Papers Proceedings), Boston, MA.

[28] D. Dutta-Roy. "Virtual meetings with Desktop", IEEE Spectrum, Vol. 35, No. 7, July 1998, pp. 47-56.

[29] Daniela E.H. Damian, Mildred L.G. Shaw, Armin Eberlein, Brian R. Gaines, Brian Woodward An Empirical Study of Facilitation of Computer-Mediated Distributed Requirements Negotiations in Fifth IEEE International Symposium on Requirements Engineering (RE'01) pp. 0128 August 2001.

[30] Darke, J , "The primary generator and the design process," Design Studies, vol. 1, no. 1, 36-44, 1979.

[31] Dasgupta, S., "The structure of design processes," in Advances in Computers, Yovits, M.C., Ed. Academic Press, 1989, pp. 1-67.

[32] David Marca,Geoffrey Bock, Groupware: Software for Computer Supported Cooperative Work (Chapter 4)., IEEE Computer Society Press.

[33] DeGrace, P. and Hulet-Stahl, L., Wicked Problems, Righteous Solutions : A Catalogue of Modern Software Engineering Paradigms. Englewood Cliffs, New Jersey: Yourdon Press, 1990.

[34] Dewan, P., Architectures for collaborative applications. In Beaudouin-Lafon, M. (Ed.), Computer Supported CooperativeWork, Trends in Software Series 7:169-193. JohnWiley & Sons, Chichester, 1999.

[35] Donald A. Norman. The Design of Everyday Things . NY. Basic books 1988.

[36] Douglas C. Engelbart, Bootstrap Institute June 1992 (AUGMENT,132811,)Toward High-Performance Organizations: A Strategic Role for Groupware.

[37] Dourish, P.,Software infrastructures. In Beaudouin-Lafon, M. (Ed.), Computer Supported Cooperative Work, Trends in Software Series 7:195-219. John Wiley & Sons, Chichester, 1999.

[38] Dubs, S. & Hayne, S. Distributed Facilitation: A Concept Whose Time Has Come? In Proc CSCW'92 Conference on Computer Supported Cooperative Work, ACM/PRESS, N.Y., 1992, pp. 314-321.

[39] E James D. Herbsleb, Audris Mockus: An Empirical Study of Speed and Communication in Globally Distributed Software Development. IEEE Trans. Software Eng. 29(6): 481-494 (2003)

[40] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. Design Patterns Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995

[41] Evert Gummesson, Quality Management in Service Organizations. New York: St. John's University and The International Service Quality Association (ISQA).

[42] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, Peter Sommerlad, Michael Stal Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. Wiley-VCH Verlag GmBH, June 2001.

[43] Freeman, P, “The nature of design,” in Tutorial on Software Design Techniques, Freeman, P. and Wasserman, A.I., Eds. IEEE, 1980, pp. 46-53.

[44] French, A. A Study of Communication and Cooperation in Distributed Software Project Teams. In Proc. International Conference on Software Maintenance. (1998), 146-154.

[45] Gable, G., "Integrating Case Study and Survey Research Methods: An Example in Information Systems," European Journal of Information Systems, (3:2), 1994, pp. 112-126.

[46] Gaoyan Xie, Yongsen Xu, Yu Li, Qian Li. Codebugger - A Software Tool for Cooperative Debugging. SIGPLAN Notices 35(2): 54-60 (2000).

[47] Gause, Donald and Gerald Weinberg, 1989, Exploring Requirements: Quality Before Design. New York: Dorset House Publishing.

[48] Gorton, I., Hawryszkiewycz, I., Ragoonaden, K., Chung, C., Lu, S., and Randhawa, G. Groupware Support Tools for Collaborative Software Engineering. In Proceedings of the 30th IEEE International Conference on System Sciences. (1997), 157-166.

[49] Gorton, I, and Motwani, S., Towards a Methodology for 24 Hour Software Production Using Geographically Separated Teams, in Proceedings of the First IFIP International Conference on Software Quality and Productivity, Hong Kong, December 5-7th, Chapman and Hall, pages 50-55, 1994.

[50] GRINTER R. E. Recomposition: Putting It all Back Together Again. In Proceedings of CSCW'98, Seattle WA, pp. 393-402, 1998.

[51] Grudin, J. (1994). Cscw: History and focus. IEEE Computer, 27(5):16-19.

[52] Gutwin, C., Roseman, M., and Greenberg, S., A usability study of awareness widgets in a shared workspace groupware system. In Proceedings of the ACM CSCW'96 Conference on Computer Supported Cooperative Work, Boston, Mass., November 16-20, 1996.

[53] Gutwin, Carl ; Greenberg, Saul: "Workspace awareness for groupware" in: Proceedings of the CHI '96 conference companion on Human factors in computing systems: common ground, Vancouver, BC, Canada, 1996, pp. 208-209.

[54] Harald Holz, Sigrid Goldmann and Frank Maurer. Working Group Report on Coordinating Distributed Software Development Projects.

[55] Herbsleb, J., and Grinter, R. Architectures, Coordination, and Distance: Conwayís Law and Beyond. IEEE Software 16, 5 (September/October 1999), 63-70.

[56] Herbsleb, J., and Grinter, R., Splitting the Organization and Integrating the Code: Conwayís Law Revisited. In Proceedings of the 1999 international conference on Software Engineering. (1999), 85-95.

[57] Hewett, T. T. (1986). The role of iterative evaluation in designing systems for usability. Proceedings of British Computer Society Human Computer Interaction Specialist Group Conference - University of York, 196-214

[58] Higa, K.Understanding Relationships Among Teleworkers' E-Mail Usage, E-mail Richness Perceptions and E-Mail Productivity Perceptions Under a Software Engineering Environment. IEEE Transactions on Engineering Management 47, 2 (May 2000), 163- 173.

[59] Hoover, S.P.and Rinderle, J.R., "Models and abstractions in design," Design Studies, vol. 12, no. 4, 237-245, 1991.

[60] IBM Online, 2003 Scalable Model-Driven Development with UML 2.0 http://www-306.ibm.com/software/sw-events/webcast/print/ R562411U95151E41.html

[61] Internet-based collection of materials for Evaluating Socio Economic Development, Final Materials December 2003 www.evalsed.info

[62] J. Kramer. Distributed Software Engineering Invited State-of-the-Art Report. Proceedings of the 16th International Conference on Software Engineering, May 16-21, 1994, Sorrento, Italy. IEEE Computer Society / ACM Press, 1994.

[63] J. Sametinger and A. Stritzinger. Exploratory software development with class libraries. In Proc. 7th Joint Conference of the Austrian Computer Society, Klagenfurt, Austria, 1992.

[64] J.Bettin. Model-Driven Software Development - An emerging paradigm for Industrialized Software Asset Development. SoftMetaWare Ltd http://web-design.ittoolbox.com/documents/document.asp?i=4335.

[65] J.F. Gilgun, "Definitions, Methodologies, and Methods in Qualitative Family Research," Qualitative Methods in Family Research. Thousand Oaks: Sage, 1992.

[66] Jacobsen, I. (1994). Object-Oriented Software Engineering. New York, Addison-Wesley.

[67] Jarvenpaa, S. L., Knoll, K., & Leidner, D. E. (1998). Is anybody out there? Antecedents of trust in global virtual teams. Journal of MIS, 14(4), 29-64.

[68] Jarvenpaa, S. L., & Leidner, D. E. (1998). Communication and Trust in Global Virtual Teams. Journal of Computer-Mediated Communication (Online at http://www.ascusc.org/jcmc), 3(4).

[69] Joey F. George, Joseph S. Valacich, J.F. Nunamaker, Jr.. “THE ORGANIZATIONAL IMPLEMENTATION OF AN ELECTRONIC MEETING SYSTEM: AN ANALYSIS OF THE INNOVATION PROCESS”.

[70] Jonathan Grudin. (1988). Why CSCW Applications Fail: Problems in the Design and Evaluation of Organisational Interfaces. Proceedings of the Conference on Computer- Supported Cooperative Work (CSCW '88).

[71] Jorn Bettin. Model-Driven Software Development An emerging paradigm for Industrialized Software Asset Development Version 0.8 June 2004

[72] Kaplan, B. and Maxwell, J.A. "Qualitative Research Methods for Evaluating Computer Information Systems," in Evaluating Health Care Information Systems: Methods and Applications, J.G. Anderson, C.E. Aydin and S.J. Jay (eds.), Sage, Thousand Oaks, CA, 1994, pp. 45-68.

[73] Kaplan, B. and Duchon, D. "Combining Qualitative and Quantitative Methods in Information Systems Research: A Case Study," MIS Quarterly (12:4) 1988, pp. 571-587.

[74] Kaplan, S.M. , “ConversationBuilder: An Open Architecture for Collaborative Work,” in Human-Computer Interaction - INTERACT '90, 1990, pp. 917-922.

[75] Karat, J.and Bennett, J., “Supporting effective and efficient design meetings,” in Human-Computer Interaction - INTERACT '90, 1990, pp. 365-370.

[76] Kent McPhee. Design Theory and Software Design. Technical Report TR 96-26 October 1996 (Revised May 1997) The University of Alberta Edmonton, Alberta, Canada.

[77] Kruchten, P. (2000). The Rational Unified Process: an Introduction. Addison-Wesley

[78] Krueger, M.W., “Environmental Technology: Making the Real World Virtual,” Communications of the ACM, vol. 36, no. 7, 36-37, Jul. 1993.

[79] Kurland, N. B. & Egan, T. D. (1999). Telecommuting: Justice and Control in the Virtual Organization. Organization Science, 10(4), 500-513.

[80] Lammers, S, Programmers at Work. Microsoft Press, 1986.

[81] Lawson, B.,How Designers Think. The Architectural Press Ltd.: London, 1980.

[82] Liam Bannon . (1993). Use, Design, and Evaluation: Steps towards an integration. Shaerding CSCW Workshop.

[83] Liam J. Bannon. CSCW: An Initial Exploration. Scandinavian Journal of Information Systems, 5:3-24, August 1993.

[84] M Wiesmann. Understanding Replication in Databases and Distributed Systems Proceedings of the The 20th International Conference on Distributed Computing Systems (ICDCS 2000) Page: 464 - 2000.

[85] M. Bauer, B. Bruegge, G. Klinker, A. MacWilliams, T. Reicher, S. Riß, C. Sandor, and M. Wagner, Design of a Component-Based Augmented Reality Framework, in Proceedings of ISAR 2001, New York, USA, 2001, IEEE Computer Society, pp. 124-133. 120

[86] MacLean, A., Bellotti, V., and Young, R., “What rationale is there in design?,” in Human-Computer Interaction - INTERACT '90, 1990, pp. 207-212.

[87] Magnus Ramage, CSCW Evaluation in Five Report CSEG/17/96, Co-operative Systems Engineering Group, Lancaster University, UK.

[88] Majchrzak, A.,Rice, R. E., King, N., Malhotra, A., & Ba, S. (2000a). Computer-mediated interorganizational knowledge-sharing: Insights from a virtual team innovating using a collaborative tool.

[89] Malone, T. W. : 1992, Experiments with Oval: A Radically Tailorable Tool for Cooperative Work, in J. Turner and R. E. Kraut (eds), Proc. of the Conf. on Computer Supported Cooperative Work CSCW'92, ACM Press, New York, pp. 289{297.

[90] Matthew Bass, Daniel Paulish Global Software Development Process Research at Siemens

[91] Meadows, C. J. (1996b). Globework: Creating Technology with International Teams (thesis)., Harvard University, Boston.

[92] Meadows, C. J.(1996a). Globalizing Software Development. Journal of Global Information Management, 4(1), 5-14.

[93] Michael Cusamano, Richard Selby, Microsoft Secrets: How the worlds most powerful software company creates technology, shapes markets, and manages people. New York Free Press, 1995.

[94] Millar, J. (1999). International Software Trade: Capability Building Through Client Relationships. A submission to: The Information Society.

[95] Mills, H. D., O'Neill, D., et all 1980. The management of software engineering. IBM Sys. J., 24(2), 414-77. (Ch. 3).

[96] Minneman, S.L. , "The Social Construction of a Technical Reality: Empirical Studies of Group Engineering Design Practice," Ph.D. thesis, Stanford University, 1991.

[97] Mostow, J.,"Toward better models of the design process," AI Magazine, vol. 6, no. 1, 44-57, Spring 1985.

[98] N. Boulila. Supporting Distributed Software Development with RD-UML Fachwissenschaftlicher Informatik-Kongress - Informatiktage 2002, Bad Schussenried: Nov 8-9, 2002. Konradin Verlagsgruppe und Stepstone AG Deutschland. Pp 107-111.

[99] N. Boulila, A. Braun, O. Creighton, T. Zhang. Tech report for SMART Technologies Inc., iBistro: Supporting Informal Meetings in Distributed Software Development. 2002.

[100] N. Boulila, A.H. Dutoit, , B. Bruegge. Group Support for Distributed Collaborative Concurrent Software Modeling. In intern. conference on Automated Software Engineering (ASE) Linz Austria 2004.

[101] N. Boulila, A.H. Dutoit, , B. Bruegge. Towards a support of Rationale-based Distributed Cooperative Group Modeling of SoftwareIn intern. conference on Automated Software Engineering (ASE) Linz Austria 2004.

[102] N.Boulila .Towards an Object-Oriented CSCW Framework for Supporting Distributed Software Modeling Fachwissenschaftlicher Informatik-Kongress - Informatiktage 2003, Bad Schussenried: Nov 8-9, 2003. Konradin Verlagsgruppe und Stepstone AG Deutschland.

[103] N.Boulila, A.Dutoit, B.Bruegge. CSCW-based Software Engineering Course: A Case Study Of Distributed Collaborative Software Modeling in

Education. Proc. Intl. Conf. on Applied Computing (Pedro Isaias, Miguel Baptista Nunes eds.), IADIS, pp. 271-278, Lisbon, Portugal, Mar. 2004..

[104] N.Boulila, A.H. Dutoit, B. Bruegge D-Meeting: an Object-Oriented Framework for Supporting Distributed Modelling of Software International Workshop on Global Software Development, International Conference on Software Engineering. Portland, Oregon, May 9, 2003.

[105] N.Boulila, A.Dutoit, B.Bruegge. Towards a Unified Object-Oriented CSCW-Framework for Supporting Distributed Group Modeling Of Software. Proc. Intl. Conf. on Applied Computing (Pedro Isaias, Miguel Baptista Nunes eds.), IADIS, pp. 613-621, Lisbon, Portugal, Mar. 2004.

[106] N.Boulila. Computer Supported Cooperative Software Engineering: A framework for supporting distributed concurrent group modeling of software. In Doctoral Consortium Proc. Intl. Conf. on Applied Computing IADIS, pp. IV11-15, Lisbon, Portugal, Mar. 2004.

[107] N.Boulila. SCOOP: A framework for supporting Synchronous Collaborative Object-Oriented Software Design Process. Cooperative Support for Distributed Software Engineering Processes ASE Linz 2004.

[108] N.Boulila. Bootstrapping Incremental Design: An Empirical Approach For Requirements Identification and Distributed Software Development. in the International Workshop on Distributed Software Development, in conjunction with 13th IEEE Requirements Engineering Conference 2005 Paris France.

[109] Nakanishi, H.,Yoshida, C., Nishimura, T., and Ishida, T. (1999). Freewalk: A 3d virtual space for casual meetings. IEEE Micro.

[110] Naur, P, “Programming as theory building,” Microprocessing and Microprogramming, vol. 15, 253-261, 1985.

[111] Nierstrasz, O. and Tsichritzis, D.: 1989, Integrated O ce Systems, in W. Kim and F. Lochovsky (eds), Object-Oriented Concepts, Databases and Applications, ACM Press and Addison- Wesley, pp. 199{215.

[112] Ocker, R. , Hiltz, S.R., Turoff, M. Fjermestad, J., Computer Support for Distributed Asynchronous Software Design Teams: Experimental Results on Creativity and Quality. In Proceedings of the 28th IEEE International Conference on System Sciences. (1995), 4-13.

[113] Osborn, A. (1957), Applied imagination: Principles and procedures of creative thinking (rev. ed.), New York: Scribner's.

[114] P. Dewan, J. Riedl. (1993) “Toward Computer-Supported Concurrent Software Engineering”, IEEE Computer, Vol. 27, pp. 17-27, Jan. 1993.

[115] Palmer, S. R. and Felsing, J. M. (2002). A Practical Guide to Feature-DrivenDevelopment. Upper Saddle River, NJ, Prentice-Hall.

[116] Parnas, D.L. and Clements, P.C., “A rational design process: how and why to fake it,” IEEE Transactions on Software Engineering, vol. 12, no. 2, 251-257, Feb. 1986.

[117] Prakash, A., Group editors. In Beaudouin-Lafon, M. (Ed.), Computer Supported Cooperative Work, Trends in Software Series 7:103-133. JohnWiley & Sons, Chichester, 1999.

[118] Rajkumar, T.,& Dawley, D. (1997). Problems and Issues in Offshore Development of Software. In L. Willcocks & M. Lacity (Eds.), Information Systems Sourcing: Theory and Practice. Oxford: Oxford University Press.

[119] Rebecca E. Grinter From Local to Global Coordination: Lessons from Software Reuse Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work . Boulder, Colorado, USA Pages: 144 - 153 Year of Publication: 2001

[120] Rebecca E. Grinter, James D. Herbsleb, Dewayne E. Perry: The geography of coordination: dealing with distance in R&D work. GROUP 1999: 306-315

[121] Rein, G.L.and Ellis, C.A., rIBIS: A real-time group hypertext system. International Journal of Man-Machine Studies, 34(3): 349-367, 1991.

[122] Robert D. Battin, Ron Crocker, Joe Kreidle Leveraging Resources in Global Software Development.

[123] Rossi, P. H , Freeman, H. E., & Lipsey, M. W. (2004). Evaluation: A systematic approach (7th ed.). Thousand Oaks, CA: Sage.

[124] Rosson, M.B , Maass, S., and Kellogg, W.A., “The designer as user: building requirements for design tools from design practice,” Communications of the ACM, vol. 31, no. 11, 1288-1298, Nov. 1988.

[125] Rudolf K. Bock, Bootstrap. 7 April 1998 http://rkb.home.cern.ch/rkb/AN16pp/node22.html

[126] S.J. Taylor and R. Bogdan, Introduction to Qualitative Research Methods. New York: John Wiley & Sons, 1984.

[127] Schlichter, Johann, Michael Koch, Martin Berger, Workspace Awareness for Distributed Teams , Proceedings of Workshop Coordination Technology for Collaborative Applications, Singapore, Wolfram Conen(ed.), 1997.

[128] Schwaber, K. and Beedle, M. (2002). Agile Software Development With Scrum. Upper Saddle River, NJ, Prentice-Hall.

[129] Schwaber, K. (1995). Scrum Development Process. OOPSLA'95 Workshop on Business Object Design and Implementation. Springer-Verlag.

[130] Scriven, Michael. "Beyond Formative and Summative Evaluation." In In M.W. McLaughlin and ED.C. Phillips, eds., Evaluation and Education: A Quarter Century. Chicago: University of Chicago Press, 1991.

[131] Simon, H.A., “The structure of ill-structured problems,” Artificial Intelligence, vol. 4, 181-200, 1973.

[132] Simon, H.A., The Sciences of the Artificial, 2 ed.. Boston, Mass.: The MIT Press, 1981.

[133] Smart Tech Inc. http://www.smarttech.com

[134] Stefik, M., Foster, G., Bobrow, D.G., Kahn, K., Lanning, S., and Suchman, L., “Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings,” in Computer-Supported Cooperative Work: A Book of Readings, Greif, I., Ed. Morgan Kaufmann Publishers, Inc., 1988, chap. 13, pp. 335-366.

[135] Stephen J, et. al. MDA Distilled - Principles of Model-Driven Architecture. Addision Wesley. Boston 2004.

[136] Stephen J. Mellor, Marc J. Balcer, Stephen Mellor, Marc Balcer Model Driven Architecture with Executable UML (TM) - Addison Wesley 2002.

[137] Steve McConnell, Rapid Development: Taming Wild Software Schedules, Microsoft Press, 1996, pp. 449-463. ISBN 1-55615-900-5

[138] Takeuchi, H., and I. Nonaka, "The new new product development game," Harvard Business Review, pp. 137-146, January-February 1986.

[139] Teege, G.: Object-Oriented Activity Support: A Model for Integrated CSCW Systems. Computer Supported Cooperative Work (CSCW): The Journal of Collaborative Computing, 5(1), pp. 93-124, 1996.

[140] Weber, R. P. (1990). Basic Content Analysis, 2nd ed. Newbury Park, CA.

[141] Willem, R.A., “Varieties of design,” Design Studies, vol. 12, no. 3, 132-136, 1991.

[142] Winograd, T, “From Programming Environments to Environments for Designing,” Communications of the ACM, vol. 38, no. 6, 65-74, Jun. 1995.

[143] Wirfs-Brock, R., Wilkerson, B., and Wiener, L., Designing Object-Oriented Software. Prentice Hall, 1990.

[144] Y.S. Lincoln and E.G. Guba, Naturalistic Inquiry. Thousand Oaks Calif.: Sage, 1985.

[145] Yin, R. K.(1994). Case Study Research: Design and Methods (Vol. 6). Newbury Park, CA: Sage.

[146] Yutaka Yamauchi, Makoto Yokozawa, Takeshi Shinohara, and Toru Ishida. Collaboration with Lean Media: How Open Source Succeeds. In Proceedings of CSCW, pages 329--338. ACM Press, 2000.