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

Andreas Braun
Accenture
Maximilianstr. 35
80539 München, Germany
andreas.braun@accenture.com

Allen H. Dutoit and Bernd Brügge
Technische Universität München
Institut für Informatik/ I1, Boltzmannstraße 3
85748 Garching b. München, Germany
{dutoit, bruegge}@cs.tum.edu

Abstract

Communication and knowledge building are challenging in distributed contexts: participants do not all know each other and work at different times and locations; the number of participants and their organization change during the project; participants belong to different communities. Hence, to deal with the global market place, it is critical to provide teams with distributed collaboration skills and tool support. To improve the collaboration in global software development (GSD), we propose iBistro [2], an augmented, distributed, and ubiquitous communication space. iBistro aims to overcome problems resulting from miscommunications and information loss in informal or casual meetings. In this paper, we specifically focus on the technical architecture for iBistro, called the *distributed concurrent blackboard architecture* (DCBA). We developed and tested an experimental prototype of the DCBA between the National University of Singapore and Technische Universität München (TUM),Munich, Germany.

Keywords: **Global software development, technological solutions, groupware.**

1 Introduction

Distributed projects leverage off tools, such as groupware, distributed repositories, and videoconferencing utilities, to accumulate and distribute knowledge and artifacts. Distributed projects, however, introduce many technical and social barriers. In addition to being geographically distributed, participants come from different corporate cultures, use different tools, follow conflicting standards, and often speak different languages. Such challenges are difficult to meet and often cause the failure of the project.

Our goal in software engineering labs at Technische Universität München (TUM) and at Carnegie Mellon University (CMU) has been to provide a realistic software engineering experience to students. We have done this by immersing students in a single, team-based, system design project to build and deliver a complex software system for a real client. Since Fall 1997, we had the opportunity to teach distributed software engineering project courses in TUM and CMU [4]. Teams of students at CMU and TUM were taught to collaborate using groupware (e.g., web sites, Lotus Notes, Email) and configuration management systems (e.g., CVS) to design and build a system for a client. Client reviews and internal reviews were conducted using videoconferencing facilities, enabling each site to present its progress and obtain feedback from the client and from the other site. While all projects were completed successfully and students acquired a number of skills for dealing with distribution, we experienced many difficulties in the areas of communication and collaboration among the sites. In particular, participants at both sites spent much more effort during solving unexpected problems and interface mismatches than would have been the case in a single site setting. Distribution made the four following obstacles especially difficult:

Inability to find stakeholders quickly. Since participants were distributed and did not know each other, finding the author of a piece of code or of a subsystem could take several days. Similarly, finding a project participant who had an area of expertise to help with a specific problem could likewise take several days.

Inability to access knowledge. Since many decisions taken by teams were taken during meetings or informal conversations, participants at the other site could not easily access the rationale of the system. Hence, participants encountered unexpected problems when enhancing or modifying components produced by the other site. While meetings were documented in meeting minutes that were available via the groupware, such records were organized chronologically and were difficult to search when looking for a specific problem.

Inability to find artifacts quickly. Even though participants used the same repository for tracking versions of their components, it was difficult to identify when new versions were checked in and which

problems new versions addressed. Similarly, a site was usually not aware of whether a new version was under test and about to be released. Consequently, sites worked often on outdated versions and produced version conflicts by solving the same problems twice.

Inability to build "group memory". During the different stages of the lifecycle, many different and altering communication media, such as email, bboards, team web pages, databases for bugs or tasks, were used. People collaborate by communicating both formally and informally. Much of the information and knowledge, however, only resides only in the people's mind or somewhere unlinked in the databases or applications; the knowledge is lost for the organization or team.

Note that all four problems noted above were caused, at least in part, by some type of communication breakdown. Researchers distinguish between informal and formal communication and recognize their application to different types of issues [8]. Formal communication is typically non-interactive and impersonal and includes, for example, formal specifications, written documentation, structured meetings. Informal communication is typically peer-oriented and interactive and includes, for example, hallway conversations, lunch breaks, and informal conversations that follow formal meetings. While formal communication is useful for coordinating routine work, informal communication is needed in the face of uncertainty and unexpected problems. Note that all three problems noted above were caused, at least in part, by lack of informal communication, typical of distributed projects [6, 8, 1].

In this paper, we describe the technical architecture for iBistro [2]. iBistto is an experimentation environment that allows distributed teams to capture, structure, and retrieve information and knowledge produced in global distributed software development projects. iBistro focuses on the integration of various sources of information, including informal meetings held in specifically equipped rooms. The technical architecture, the *distributed concurrent blackboard architecture* (DCBA) for iBistro has been implemented and evaluated during a distributed project at the National University of Singapore and TUM.

This paper is structured as follows. Section 2 provides an overview of iBistro and details how iBistro can be used to capture, structure, and retrieve knowledge, and more generally, address problems such as finding stakeholders, accessing knowledge, finding artifacts, and building a group memory, as we identified above. Section 3 lists our results achieved so far and concludes this paper by outlining the outlook for iBistro.

2 The iBistro System

We start our overview of iBistro's architecture and design by recapitulating the specific characteristics of both (informal) meetings and software engineering to substantiate our design decision for the blackboard model [7] as the chosen architectural pattern.

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

In other words, the process of building software is reminiscent of the process of building up a wall with little stones "step-by-step". It can be seen as *knowledge assembly*, in contrast to *search for solutions*. This view of the problem domain suggests a blackboard-based approach. The blackboard model is originally used in opportunistic problem-solving to deal with non-computable and diverse problems in AI. In this section, we introduce our modified blackboard which we call a *distributed concurrent blackboard architecture* (DCBA).

The DCBA is an approach to deal with the variety of events and context that occur during (informal) meetings [3]. In our architecture, various *capture components* record contextual events (such as people entering or leaving the meeting room or tracking of electronic events), track discussions and bboard activity, as well as audio, video, and whiteboard content. Several specialized *knowledge sources* seize the captured events to create new types of information or knowledge, for instance a hypothesis (for instance argumentation or intermediate work products; not part of the deliverables) or a solution (e.g., artifacts that are part of the deliverables). This process of knowledge construction creates an abstraction hierarchy of knowledge that is stored in several *layers of abstraction* within the blackboard. Figure 1 displays the simplified overview of the iBistro system.

The single distributed concurrent blackboard space is build up from local blackboard installations (e.g., "Munich" and "Singapore"). The distributed concurrent blackboard provides a transparent means for knowledge sources and users to access one single portal, regardless of its actual technical implementation.

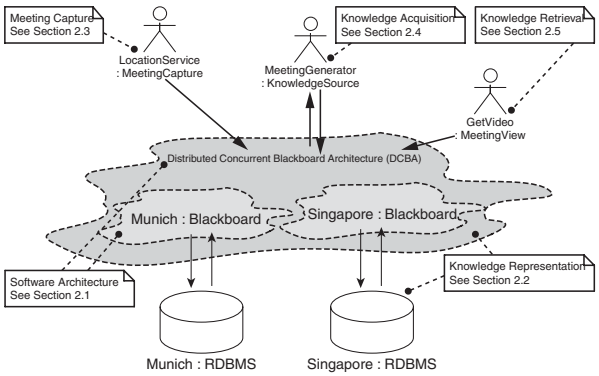

Figure 1: Overview of the iBistro System.

Figure 1 displays five fundamental concepts for iBistro. In the next sections, the design of iBistro and the usage of these concepts are detailed as described in the following overview.

2.1 Software Architecture

The *distributed concurrent blackboard* serves as a primary data repository. It is a virtual portal for all users and components which is made from many interconnected and local blackboards. Each local blackboard, however, provides transparent access for local tools to the complete knowledge base. Hence, tools do not need to know where the data are stored. In Figure 1 for example, blackboard systems at Munich and Singapore together form the global DCBA.

Small data objects are completely replicated over time, enabling faster access and concurrent access. Larger data objects, such as video files or other artifacts, are not completely replicated but accessed directly in smaller chunks. This saves blackboard storage space and transfer time.

The blackboard is *concurrent* in that several components can access the repository at the same time, either using read or write operations. This also implies that concurrent users or components likely will create several concurrent *versions* of a data item (see next section for more information on version control).

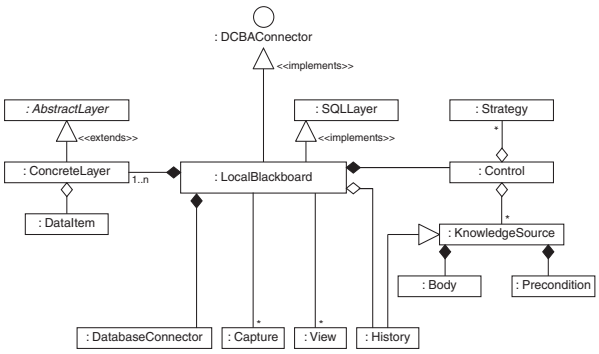

Figure 2: The DCB Architecture in Detail.

Figure 2 shows a class diagram representing one single iBistro system. The following components relate to their corresponding principles in iBistro: The **blackboard** (including the **layers** and **database**) serves as a global data structure for knowledge representation and storage. **Control** and **strategy** components orchestrate the overall knowledge acquisition process by directing the knowledge sources (KS). The **knowledge sources** achieve the knowledge building process on a certain level of abstraction. The **precondition** of a KS implements a rule for the control component to decide whether to execute the **body** (the actual code) of the KS. **Capture** and **view** components access the repository without being directed by the control and simply write data items to the repository or read from out to present the information to the users.

2.2 Knowledge Storage & Representation

The blackboard is a global data structure and serves as a medium for all communication within the architecture. The blackboard can store any kind of information, called *data item*. However, the information stored is categorized into several *levels of abstraction*, which are stored in certain *layers* in the blackboard repository. The blackboard is designed flexible to allow for different representation schemes and organization of layers. For now, we assume that the blackboard contains three types of data items, represented in three layers accordingly:

Layer *n*:	Solutions and decisions.
Layer *n-1*:	Hypothesis and partial solutions.
Layer *0*:	Context and raw data.

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

The layers represent the concept of knowledge acquisition: data is transformed to information, which is transformed to knowledge. The layers also represent the search for solutions, as explored alternatives, regardless of their later use in a solution, are considered as an important contribution and called *hypothesis*. All possible hypothesis and solutions that might be derived from a given set of data items in layer 0 are called the *solution space*. For an iBistro meeting, the solution space would consist of all possible interpretations of a meeting between a certain set of individuals. However, many of

these potential solutions are not found by the system. Instead of that, the system might identify a subset of the solution space.

The elements of each layer of the blackboard are composed of elements of the layer below or from the same layer. For instance, layer 0 might contain a bitmap of a whiteboard snapshot depicting the topic and agenda of a meeting. A specific knowledge source, such as an OCR component, then might be used to interpret the bitmap and create a machine-readable object, such as a text-object. The resulting object typically is stored within the same or one level higher layer than the source object (*bottom-up analysis*), or vice-versa (elements at lower layers are created from higher-level objects, *top-down analysis*). In top-down analysis, for instance, a changed line in the source code could be linked back to the related position in the source inspection meeting video.

In addition to the organization in layers and in contrast to existing blackboard systems, all data items also may exist in several versions. Similar to version control in software development, two knowledge sources may create two concurrent successors of a data item. Each of them later can be reconsidered or dropped independently.

2.3 Information and Meeting Capture

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

2.4 Knowledge Acquisition

Knowledge sources, pick up basic data items and work on them, potentially by using and combining the information captured by several different meeting capture components. A knowledge source is responsible for knowledge acquisition at a certain level of abstraction. Knowledge sources closely conform to the related concept introduced with generic blackboard architectures. Most of the rules introduced there do apply in the DCBA as well: knowledge sources exclusively work with information stored on the DCBA. They are only able to communicate with other knowledge sources using the DCBA. A knowledge source typically works on information from a specific layer and rises the information to the next level higher.

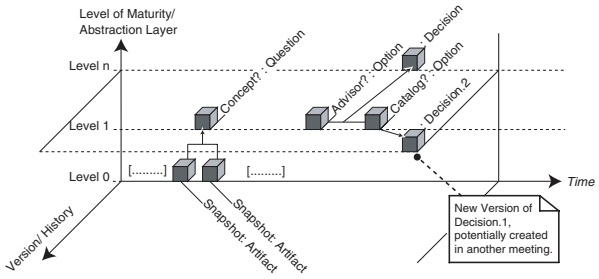

Figure 3: Knowledge in the DCBA seen as a 3D-model.

In meetings in iBistro, the MINUTEGENERATOR is a specific knowledge source to work on the diverse information created during a informal meeting, including the recorded video stream, and to create linkage of knowledge with external data items (e.g., knowledge from other meetings or data items in general). The tool is used exclusively by a dedicated *meeting champion* after the meeting. The MINUTEGENERATOR implicitely creates indices which are used by the user to navigate through the knowledge space. Knowledge is linked accordingly to its logic cohesion and by version. By logic cohesion we understand that, for instance, a question rised by one person is logically linked to that originator; later answers to the question are also interlinked with the question. Alternate versions of the question, for instance rephrased or more precise versions of the questions, are linked as a new version to the initial question. This builds up a complex network of knowledge, dependencies, and versions.

After the post-mortem processs, the hierarchy of information and knowledge now stored in the meeting minutes can be translated to a three-dimensional model as shown in Figure 3. The three axes represent the timeline (x-axis), level of abstraction (represented in the blackboard layer, y-axis), and version or knowledge-interlinkage (z-axis).

2.5 Knowledge Retrieval and Navigation

Knowledge views provide access to the contents and structure of the knowledge space. Similar to the model-view-controller paradigm [5], a variety of knowledge views might provide different visualizations of the state of a knowledge space. Knowledge views are used in particular to provide a human-computer-interface (HCI) to the information and knowledge stored in a knowledge space.

The meeting minutes stored in the repository represent the natural flow of the meeting, including external artifacts, events, or annotations from other sites or an individual's personal computer. A self-evident way to view such a meeting is to playback the meeting as a mul-

timedia archive, thus enabling non-participants to access the raw data. In iBistro, the SMIL MEETINGVIEWER generates on-demand a SMIL[1] [9] file (or data stream) to represent the meeting along with the captured requirements, context, argumentation, and so on. This allows interested people to navigate through a meeting using any SMIL compliant video player, such as RealPlayer™ or Quicktime to view the meeting. As the content of the meeting follows a common timeline, the 'Clip Position' slider is used to navigate through the captured audio, video, as well as other content such as requirements. Alternatively, the history events can be used to jump to specific segments of the meeting minutes, for example, navigating an option will move the position slider to the frame where the option was first suggested. Graphical views of requirements or rationale can be displayed using HTML or by generating bitmaps on demand.

Displaying the multi-dimensional structure of knowledge, such as context-links between stored entries which allow navigation, is non-trivial. Thus, a specific 3D-meeting view facilitates the *n*-dimensional navigation through the captured knowledge from various sites.

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

3 Status and Conclusion

The iBistro DCBA has been developed and evaluated in a distributed setting between the National University of Singapore and TUM. This setup revealed some technical difficulties and deficiencies, especially regarding "live" audio and video quality due to limited bandwidth and camera orientation problems. This shows the importance of local post-meeting processing and information structuring, as communication then is based on the electronic meeting minutes. The distributed setup also showed the strengths of iBistro compared to simpler electronic communication (such as email).

The DCBA builds a rich *group memory* by integrating artifacts and surrounding information and knowledge (rationale, stakeholders, ...) information into a common knowledge space. This allows participants to:

- *Find stakeholders* by looking at related efforts, discussions, or material.
- *Access knowledge* by browsing the linked structure of the knowledge space.
- *Find artifacts quickly* by issue, stakeholder, topics, location, or any other node in the knowledge space.
- Understand and learn from the *history* of the ongoing project or former projects by seeing rationale entries, which include argumentation, alternatives, and decisions.

In this paper, we motivated the need to integrate various sources of information and knowledge, including informal communication, in GSD. We illustrate how a group memory is build from various knowledge sources. We propose to address some of the issues surrounding informal communication by supporting the efficient capture, structure, and navigation of meeting minutes and their integration into the long term project memory embedded in tools and documents. We described the distributed concurrent blackboard architecture as a connecting technical architecture to achieve these goals. We finally introduce our experimental environment used and conclude by presenting our current status.

[1] SMIL™ enables simple authoring of multimedia presentations over the Web. A SMIL presentation can be composed of streaming audio, streaming video, images, text or any other media type.

References

[1] A. Al-Rawas and S. Easterbrook. Communication problems in requirements engineering: A field study. In *Proc. First Westminster Conf. Professional Awareness in Software Engineering*, Univ. Westminster, London, 1996.

[2] A. Braun, B. Bruegge, and A. H. Dutoit. Supporting informal requirements meetings. In *7th International Workshop on Requirements Engineering: Foundation for Software Quality. (REFSQ'2001)*, volume 7, Interlaken, Switzerland, June 2001.

[3] A. Braun, B. Bruegge, A. H. Dutoit, T. Reicher, and G. Klinker. Experimentation in context-aware applications. Submitted to HCI Journal for publication in special issue on context-aware computing, 2000.

[4] B. Bruegge, A. H. Dutoit, R. Kobylinski, and G. Teubner. Transatlantic project courses in a university environment. In *7th Asia-Pacific Software Engineering Conference*, Singapore, Dec. 2000. APSEC.

[5] S. Burbeck. Application programming in Smalltalk-80: How to use Model-View-Controller (MVC), 1987.

[6] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. In *Communications of the ACM*, volume 31(11), Nov. 1988.

[7] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy. The hearsay-ii speech-understanding system: Integrating knowledge to resolve uncertainty. In *ACM Computing Surveys*, volume 12 (2), pages 213–253, 1980.

[8] R. Kraut and L. Streeter. Coordination in software development. In *Communications of the ACM*, volume 38(3), Mar. 1995.

[9] W3C. SMIL. Technical report, World Wide Web Consortium, 1998.