CHAPTER 2
THEORITICAL BACKGROUND

This chapter will mention some theoretical notions in computer engineering that are connected with this thesis work. Those will include the theory of synchronous groupware in computer supported collaborative work (CSCW), development process in software engineering that the system wants to cover, and distributed systems issues. Further, this chapter will also mention some available technology and tools to develop a synchronous groupware, as well as some examples of synchronous groupware available.

2.1. Fundamental Issues of Synchronous Groupware for CSCW

Groupware is defined by Ellis, Gibbs and Rein [5] as "computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment". The concept common task and a shared environment are fundamental in this definition. Thus, this excludes multi user systems whose users may not share a common task. The definition does not specify that the users be active simultaneously. Groupware that specifically supports simultaneous activity is called real-time groupware; otherwise, it is non-real-time groupware [5].

There are some basic/general aspects/characteristics of a groupware system for Computer Supportive Collaborative Work (CSCW) described by Terzis et al.[24]: communication strategy, coordination, distribution, scalability, openness, web exploitation and some non-functional characteristics.

2.1.1. Communication Strategy

Communication strategy refers to different ways of exchanging information. There are two communication strategies regarding the time of communication:

synchronous and asynchronous. Some of the groupware applications support only synchronous (e.g. audio and video conferencing), some only support asynchronous, and some both.

Groupware can be differentiated in time and space considerations as shown in Figure 1, while this thesis will focus on the same time different place area, namely Synchronous Distributed Interaction.

Furthermore, synchronous communication can be differentiated into loosely coupled and tightly coupled. Tightly coupled synchronous groupware implement the principle WYSIWIS (What You See Is What I See). In this principle, every user will have the exactly same view of the software using a shared focus, or simply translated as seeing the same computer screen together. There is another improvement of this principle, which is relaxed WYSIWIS, where users in the same session might have a different view of the artifact although the information is just the same.

		TIME	
		Same Time	Different Time
SPACE	Same Place	Face-to-face interaction (Spontaneous collaborations, formal meetings, classrooms)	Asynchronous interaction (Design rooms, Project scheduling)
	Distributed	Synchronous distributed Interaction (Video conferencing, net meeting, phone calls)	Asynchronous distributed Interaction (Emails, writing, voice mails, fax)

Figure 1. Time Space Taxonomy [5]

2.1.2. Coordination

Coordination refers to the organization of user interactions. Different group processes require different coordination policies, from flexible ones (e.g. brainstorming session) to rigid ones (e.g. workflow automation) [24].

2.1.3. Distribution

Distribution refers to the physical location of the group members. The distribution policies manage from physically collocated, to worldwide distributed groups [24].

2.1.4. Scalability

Scalability refers to the different group sizes that an application supports. It can range from a few users (e.g. email) to a few thousands of users (e.g. web conferencing) [24].

2.1.5. Openness

Openness refers to the flexibility of a system in integrating other applications, working on different hardware platforms, and the use of different concurrency control and awareness mechanisms [24].

2.1.6. Web exploitation

The Web provides a shared information space and supports basic collaboration. Its success, in combination with the usual failures of groupware systems, led to it being considered as an appropriate basis for groupware applications [24]. Thus exploitation of the Web becomes an important aspect of groupware [24].

2.1.7. Additional non-functional characteristics

Other non-functional characteristics include things like fault-tolerance, security, safety, integrity, etc [24]. The support of these characteristics and their flexibility differs significantly in the various applications even within the same category [24].

2.2. Common Functionalities/Application of CSCW

This section describes several types of groupware applications and their associated design options. Comparing those design options across applications yields interesting new perspectives on well-known applications. Also, in many cases, these systems can be used together, and in fact, are intended to be used in conjunction. For example, group calendars are used to schedule videoconferencing meetings, multi-player games use live video and chat to communicate, and newsgroup discussions spawn more highly-involved interactions in any of the other systems.

2.2.1. Asynchronous groupware

Below are some means of applications that are categorized as asynchronous groupware applications [www8]:

- **Email** is by far the most common groupware application (besides of course, the traditional telephone). While the basic technology is designed to pass simple messages between 2 people, even relatively basic email systems today typically include interesting features for forwarding messages, filing messages, creating mailing groups, and attaching files with a message. Other features that have been explored include: automatic sorting and processing of messages, automatic routing, and structured communication (messages requiring certain information).
- **Newsgroups** and **mailing lists** are similar in spirit to email systems except that they are intended for messages among large groups of people instead of 1-to-1 communication. In practice the main difference between newsgroups and mailing lists is that newsgroups only show messages to a user when they are explicitly requested (an "on-demand" service), while mailing lists deliver messages as they become available (an "interrupt-driven" interface).
- **Workflow systems** allow documents to be routed through organizations through a relatively-fixed process. A simple example of a workflow

application is an expense report in an organization: an employee enters an expense report and submits it, a copy is archived then routed to the employee's manager for approval, the manager receives the document, electronically approves it and sends it on and the expense is registered to the group's account and forwarded to the accounting department for payment. Workflow systems may provide features such as routing, development of forms, and support for differing roles and privileges.

- **Hypertext** is a system for linking text documents to each other, with the Web being an obvious example. Whenever multiple people author and link documents, the system becomes group work, constantly evolving and responding to others' work. Some hypertext systems include capabilities for seeing who else has visited a certain page or link, or at least seeing how often a link has been followed, thus giving users a basic awareness of what other people are doing in the system—page counters on the Web are a crude approximation of this function. Another common multi-user feature in hypertext (that is not found on the Web) is allowing any user to create links from any page, so that others can be informed when there are relevant links that the original author was unaware of.
- **Group calendars** allow scheduling, project management, and coordination among many people, and may provide support for scheduling equipment as well. Typical features detect when schedules conflict or find meeting times that will work for everyone. Group calendars also help to locate people. Typical concerns are privacy (users may feel that certain activities are not public matters), completeness and accuracy (users may feel that the time it takes to enter schedule information is not justified by the benefits of the calendar).
- **Collaborative writing systems** may provide both realtime support and non-realtime support. Word processors may provide asynchronous support by showing authorship and by allowing users to track changes and make annotations to documents. Authors collaborating on a document may also be given tools to help plan and coordinate the authoring process, such as

methods for locking parts of the document or linking separately-authored documents. Synchronous support allows authors to see each other's changes as they make them, and usually needs to provide an additional communication channel to the authors as they work (via videophones or chat).

2.2.2. Synchronous groupware

Synchronous application in a groupware basically means that a single application can be used in a multi user environment as if the application runs for a single user. There are some synchronous applications categories [www8]:

- **Shared whiteboards** allow two or more people to view and draw on a shared drawing surface even from different locations. This can be used, for instance, during a phone call, where each person can jot down notes (e.g. a name, phone number, or map) or to work collaboratively on a visual problem. Most shared whiteboards are designed for informal conversation, but they may also serve structured communications or more sophisticated drawing tasks, such as collaborative graphic design, publishing, or engineering applications. Shared whiteboards can indicate where each person is drawing or pointing by showing telepointers, which are color-coded or labeled to identify each person.
- **Video communications** systems allow two-way or multi-way calling with live video, essentially a telephone system with an additional visual component. Cost and compatibility issues limited early use of video systems to scheduled videoconference meeting rooms. Video is advantageous when visual information is being discussed, but may not provide substantial benefit in most cases where conventional audio telephones are adequate. In addition to supporting conversations, video may also be used in less direct collaborative situations, such as by providing a view of activities at a remote location. The Usability First site

maintains a bibliography of papers on the user interface design of video communications systems.

- **Chat systems** permit many people to write messages in real-time in a public space. As each person submits a message, it appears at the bottom of a scrolling screen. Chat groups are usually formed by having listing chat rooms by name, location, number of people, topic of discussion, etc. Many systems allow for rooms with controlled access or with moderators to lead the discussions, but most of the topics of interest to researchers involve issues related to un-moderated realtime communication including: anonymity, following the stream of conversation, scalability with number of users, and abusive users. While chat-like systems are possible using non-text media, the text version of chat has the rather interesting aspect of having a direct transcript of the conversation, which not only has long-term value, but allows for backward reference during conversation making it easier for people to drop into a conversation and still pick up on the ongoing discussion.
- **Decision support systems** are designed to facilitate groups in decision-making. They provide tools for brainstorming, critiquing ideas, putting weights and probabilities on events and alternatives, and voting. Such systems enable presumably more rational and even-handed decisions. Primarily designed to facilitate meetings, they encourage equal participation by, for instance, providing anonymity or enforcing turn-taking.
- **Multi-player games** have always been reasonably common in arcades, but are becoming quite common on the internet. Many of the earliest electronic arcade games were multi-user, for example, Pong, Space Wars, and car racing games. Games are the prototypical example of multi-user situations "non-cooperative", though even competitive games require players to cooperate in following the rules of the game. Games can be enhanced by other communication media, such as chat or video systems.

2.3. Awareness in Synchronous Groupware

"Awareness" plays an important role in CSCW systems. Without awareness, there would be no collaboration. The basic meaning of awareness in collaborative systems is to know who the collaborators are, what they are doing, what they have done and what they want to do [9]. Hence, it is also called group awareness. With group awareness, the collaborators have a basic idea of his or her collaborating working environment that he/she is using and contributing.

Traditional CSCW systems apply four kinds of interleaved awareness [9] :

- Workspace awareness is about who is present in the workspace, where they are, and what they are doing;
- Informal awareness means that the general sense of who is around and what they are depending on;
- Social awareness is the information that a person maintains about others in a social or conversational context; and
- Group-structured awareness involves knowledge about such things as people's roles and responsibilities, their positions on an issue, their status, and group process.

2.3.1. WYSIWIS [9]

WYSIWIS means What You See Is What I See. This is the aim of most synchronous CSCW systems. Synchronous systems do not only want to support the same views to the shared documents but also the shared views of each collaborator, because in a face-to-face environment, all the collaborators share the views to the environment.

WYSIWIS is one of the prevalent approaches in designing systems to support synchronous interaction and cooperation, that is, it provides the users with all their individual and familiar tools. WYSIWIS emphasizes that collaboration requires the commonalities among the collaborators.

Evidently, traditional CSCW systems emphasize more on WYSIWIS then WYSINWIS. WYSIWIS is the aim of explicit awareness. The problem is that we still need to group awareness when WYSIWIS cannot be practically applied. Therefore, relaxed WYSIWIS, What You See is What I think You See (WYSIWITYS, and What You See is Not What I See (WYSINWIS) are proposed to the CSCW world. They are all mechanisms that can be used to support implicit awareness.

2.3.2. Relaxed WYSIWIS [9]

WYSIWIS enforces strict synchronization of different user views onto a shared workspace. Relaxed WYSIWIS coupled with techniques for promoting multi user awareness and concurrency control mechanisms for interleaving users actions have focused on enabling the possibility of collaboration while retaining a high degree of individual autonomy. WYSIWIS supports that different users at different displays were forced to see the same part of a virtual workspace. Relaxed WYSIWIS was led by a less strictly coupled approach where different user's views could diverge.

Although the WYSIWIS idealization recognizes that efficient reference to common objects depends on a common view of the work at hand, WYSIWIS was found to be too limiting and relaxed versions were proposed to accommodate personalized screen layouts.

Relaxed WYSIWIS actually means that groupware should not always support full or strict WYSIWIS principles at all time and all places, it should have flexible choices for users to choose.

2.3.3. WYSINWIS [9]

WYSINWIS stands for What You See Is Not What I See. It is terminology to emphasize the real collaboration among people in a group take different responsibilities and have different rights.

In reality, every person has his or her view to the world. They ask, serve, and interact with the world in their different ways. WYSINWIS definitely does not want to support the explicit awareness, but it still can support collaboration in an efficient way. WYSINWIS will use the implicit awareness to help collaborators to contribute in a collaborative work. WYSINWIS emphasizes that one user's view is not totally the same as others' view on shared objects.

Actually, WYSINWIS touches another extreme point that emphasizes the personalized view instead of shared view and role-based collaboration is the direct way to support it.

- WYSIWIS emphasizes commonalities requirements in collaborative environments and WYSINWIS emphasizes the personalities in the working environments.
- WYSIWIS emphasizes the benefits of face-to-face meeting and brainstorming for creative collaboration and WYSINWIS emphasizes the benefits of efficient working when collaborators have clear rights and responsibilities.
- WYSIWIS still has a long way to be satisfied by the users and WYSINWIS has just begun noticed and has many potential applications.
- WYSIWIS coupling has two main disadvantages. It can lead to wars whenever users are forced to share changes they do not want to share. WYSIWIS requires the overhead of communicating all user changes to all of the other users, which is inefficient, especially when the users do not wish to share some of these changes.
- WYSINWIS does not have these disadvantages but does not have any of the advantages of WYSIWIS either.

2.4. Collaborative Architecture

In software architecture, client and server architecture model distinguishes client systems from server systems, which communicate over a computer network

[www7]. A client-server application is a distributed system comprising both client and server software [www7]. A client software process may initiate a communication session, while the server waits for requests from any client [www7].

The distribution of the server and client basically can be differentiated into 2; server based or centralized architecture and peer-to-peer or replicated architecture. Between these 2 extremely different architectures, there is a hybrid or semi-replicated architecture which combine both.

In client server component based software, architecture will determine the nature of the components of the system and the placement of these components of various users participating in collaborative session.

2.4.1. Centralized

In this architecture model, all users interact with one single program, which resides on one of users' workstation, as the only server. The program processes all inputs and distributes the output to all users.

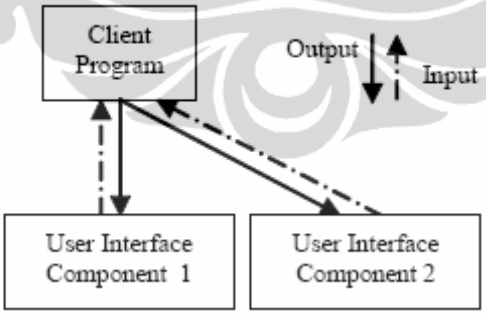

Figure 2. Centralized Architecture [1]

This architecture model is better usage when [1]:

- The computer on which the central program component executed is more powerful than the other computer participating in the collaboration

- The central site is connected by fast network connections to all others site
- The user at the central site provides a proportionately large amount of input
- The numbers of users is large

2.4.2. Replicated

In this architecture model, the program is replicated on all of the users' workstations and has the user interact with the local program replica. Therefore, there are 1 user part and 1 server part in a single client workstation. This architecture synchronizes the program replicas by broadcasting each user's input to all of them.

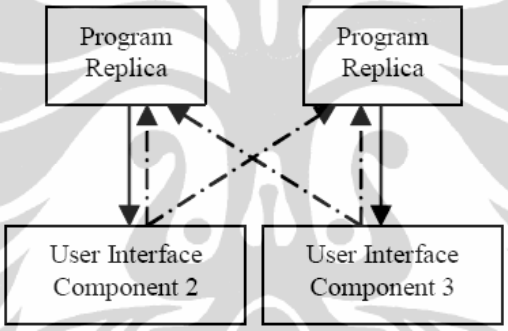

Figure 3. Replicated Architecture [1]

The basic advantage [1]:

- The replicas need not to share the complete state if programmer-defined code is defined to multicast selected input events.

While the drawbacks are [1]:

- Each replicated operation must be idempotent (multiple executions equivalent to single execution)
- Floor control is needed to ensure consistency (though in some cases application specific operation can be used instead)

2.4.3. Hybrid (Semi-Replicated)

In this architecture model, the program can be replicated and also serve multiple user interface component. Therefore there will be fewer servers than clients. All servers will have to synchronize each other, while a client will only have to communicate with its server.

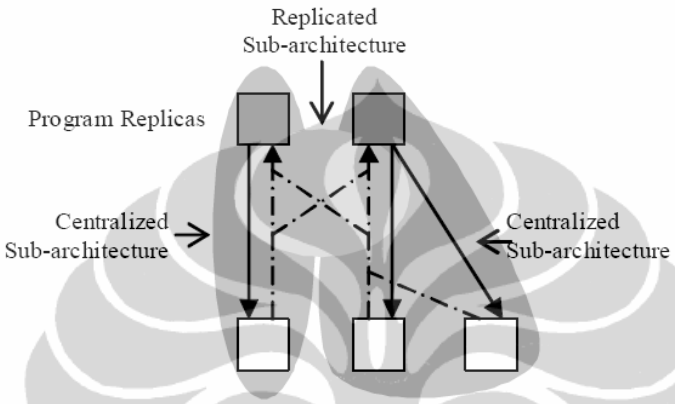

Figure 4. Hybrid/Semi-Replicated Architecture [1]

Hybrid architecture can be varied into 2 [1]:

- Static: the architecture is fixed, who owns the replica and who should serve he user interface
- Flexible/Dynamic
  - Flexible at start-up. At start up, before start the users decide the configuration.
  - Flexible at run-time: if in the middle of session, users want to reconfigure the architecture or the condition forced to do that.

Dynamic architecture is needed in pervasive collaboration (independent of network, location, device, application used). In flexible at run time, for example, when a user joins or leaves a session, some conditions to be considered in changing the architecture are:

- Network connection
- Computer quality (powerful or not)
- Distribution of users

- Number of users

2.5. Component Based Groupware Models

Component-based software is aimed to increase the reusability and interoperability of software. Component-based development aims at constructing software artifacts by assembling prefabricated, configurable and independently evolving building blocks, the so-called components [22]. Components are binary, self-contained and reusable building blocks providing a unique service that can be used either individually or in composition with the service provided by other components [22]. A software component can be deployed independently and is subject to composition by third parties [3].

Other definition comes from D'Souza and Wills [2], components is a "unit of software that encapsulates its design and implementation and offers interfaces to the outside, by which it may be composed with other components to form a larger whole."

The common (traditional) object-oriented software development aims at providing reusability of object type definitions (object classes) at design and implementation levels [3]. In contrast, component-based development aims at providing reusability of components at deployment level [3]. Therefore, components in component-based software development will represent functionality pieces that are ready to be installed and executed in multiple environments [3].

Components are most useful if there is no need at all to understand their internal program. If a component's implementation is fully hidden and all composition is strictly based on the specification of the component's interfaces, then the component can be replaced by a newer version or by one from a different source. It is important that components come in a binary form ready for use, and not in the form of a traditional machine-specific executable. To be automatically composable, a component should come with machine-readable information on which interfaces it uses [3].

Components, if seen as units that are acquired, deployed, and dynamically loaded and linked, are rather like classes instead of objects. In theory it is possible to unify objects and classes. Indeed, this has been done in a number of experimental systems. If such objects are made persistent and self-contained (including all required implementation facets), they can serve as components. In practice, this unification leads to significant problems. The sharing of a single class by many objects introduces the advantage of maintainability and consistency. However, even classes are usually too tightly coupled with certain other classes to be useful components. It is thus useful to group a set of related classes and possibly further resources into a component. Instead of synthesizing new code to compose components into larger units, it is often more useful to capture the composition using objects—instances of classes that are carried by the used components. Such a composition needs to be made persistent; the persistent objects form part of the resources that come with a component.

Some issues to be considered for the design of component based groupware are [19]:

- How to recognize the likely requirements for extensibility and composability that guide the design of component groupware architecture? Component groupware should permit adjustment of systems depending on the people to support, the task they perform, and the context in which the task is performed. Similarly, when any of these factors evolves over time, the supporting system should be able to be evolved along with it.
- Which types of components should be identified in component groupware architecture and how should these types of components interact?
- Which mechanisms should be supported to compose a groupware system from components? How to support composition by end-users as a form of tailor-ability and how to support composition by programmers (during design time, as a form of engineering)? Do we need different component groupware architectures for these different types of composition? E.g., a distinction similar to the distinction between Lego Duplo (large, easily connectable meaningful bricks for small children, versus Lego Classic and

Lego Technic (small, more intricately connectable abstract bricks for older engineering oriented children)?

- Which mechanisms (if at all) should be supported to *extend* a component-based groupware system. Extension with components from various vendors/parties introduces the requirements of third party composition, i.e., less opportunity to rely on integration testing to obtain a properly behaving system. This is most severe when the composition is done by end-users (e.g. click on "install new component" is all that is required from the end-user?),
- How to use existing generic component-based platforms (such as JavaBeans, COM+ and CORBA component model) and how should they be extended in order to support engineering of component-based groupware?
- How do design component groupware architectures that *abstract* from distribution (and focus on intrasystem composability and interaction, typically specified in terms of interfaces, calls and events). How to design component groupware architectures that *exhibit* distribution (and focus on inter-system interoperability and interaction, typically specified in terms of protocols)? In the latter case, how does one decide, based on e.g. requirements on response times and consistency, when and where to apply a distributed, replicated or centralized architecture?
- How to design component groupware architectures primarily aimed at supporting synchronous communication, versus those primarily aimed at supporting asynchronous communication? How to combine them? Real-time demands and consistency problems create extra requirements for systems that support synchronous interaction. Record and replay facilities (components?) may be needed in systems that have to support both synchronous and asynchronous forms of interaction.
- Should component groupware architectures that include support for synchronous interaction be targeted towards loosely-controlled

participation (MBone-style) tightly-controlled participation (Microsoft NetMeetingstyle) in conferences, or both?

2.5.1. CORBA [www6]

CORBA is the acronym for Common Object Request Broker Architecture, OMG's open, vendor-independent architecture and infrastructure that computer applications use to work together over networks. Using the standard protocol IIOP, a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network.

CORBA applications are composed of objects, individual units of running software that combine functionality and data, and that frequently (but not always) represent something in the real world. Typically, there are many instances of an object of a single type—for example, an e-commerce website would have many shopping cart object instances, all identical in functionality but differing in that each is assigned to a different customer, and contains data representing the merchandise that its particular customer has selected. For other types, there may be only one instance. When a legacy application, such as an accounting system, is wrapped in code with CORBA interfaces and opened up to clients on the network, there is usually only one instance.

The IDL interface definition is independent of programming language, but maps to all of the popular programming languages via OMG standards: OMG has standardized mappings from IDL to C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python, and IDLscript.

This separation of interface from implementation, enabled by OMG IDL, is the essence of CORBA - how it enables interoperability, with all of the transparencies the creator claimed. The interface to each object is defined very strictly. In contrast, the implementation of an object - its running code, and its data - is hidden from the rest of the system (that is, encapsulated) behind a boundary that

the client may not cross. Client's access objects only through their advertised interface, invoking only those operations that that the object exposes through its IDL interface, with only those parameters (input and output) that are included in the invocation.

In CORBA, every object instance has its own unique object reference, an identifying electronic token. Clients use the object references to direct their invocations, identifying to the ORB the exact instance they want to invoke (Ensuring, for example, that the books you select go into your own shopping cart, and not into your neighbor's.) The client acts as if it's invoking an operation on the object instance, but it's actually invoking on the IDL stub which acts as a proxy. Passing through the stub on the client side, the invocation continues through the ORB (Object Request Broker), and the skeleton on the implementation side, to get to the object where it is executed.

The CORBA Component Model (CCM) provides a platform and language independent component model. Although the standard model is still a work-in-progress it defines e.g., mechanisms to discover the capabilities of a component, to define provided and used interfaces, and an event mechanism. CCM components are designed to be interoperable with Enterprise JavaBeans.

2.5.2. JavaBeans [6]

JavaBeans is Java's component model. It allows users to construct application by piecing components together either programmatically or visually (or both). The support of visual programming makes component-based software development truly powerful. The JavaBeans model is made up of architecture and API. Together, these elements provide a structure so that components can be combined to create an application. Components are provided with tools necessary to work in the environment, and they exhibit certain behaviors that identify them as such. One important aspect of this structure is containment. A container provides a context in which components can interact.

JavaBeans provide class and interface discovery as a mechanism to locate a component at runtime and to determine its supported interfaces so that these interfaces can be used by others. The component model also provides a registration process for components to make itself and its interfaces known. Dynamic (or late) binding allows components and applications to be developed independently. An application does not have to include a component in the development process in order to use it at runtime; it only needs to know what a component is capable of doing. Dynamic discovery also allows developers to update components without having to rebuild the application that use them. This discovery process can also be used in design-time environment. The development tool may be able to locate a component and make it available for use by the designer.

An event is something of importance that happens at a specific point in time. An event can take place due to user action or initiated by other means. Components will send notifications to other object interest of that event.

Components must be able to participate in their container's persistence mechanism so that all components in the application can provide application-wide persistence in a uniform way. This would be very important if reuse is one of the developer concerns.

The component environment allows the individual components to control most of the aspects of their visual presentation. The component is free to choose its visual presentation, but layout should be design effectively as it is an important aspect in visual presentation. The container and the components work together to provide a single application that presents itself in a uniform fashion. The application appears to be working as one unit, even though within the components development model, the container and the components probably have been developed separately by other developer.

The following is the advantages of using JavaBeans:

- Compact and easy components

- Portable. As it is built purely in Java, JavaBeans portal to any platform that supports Java runtime environment
- Leverages the strengths of Java platform
- Flexible build-time component editors

2.5.3. DCOM [www4]

Microsoft COM (Component Object Model) technology in the Microsoft Windows-family of Operating Systems enables software components to communicate. COM is used by developers to create re-usable software components, link components together to build applications, and take advantage of Windows services. The family of COM technologies includes COM+, Distributed COM (DCOM) and ActiveX Controls.

COM is used in applications such as the Microsoft Office Family of products. For example COM OLE technology allows Word documents to dynamically link to data in Excel spreadsheets and COM Automation allows users to build scripts in their applications to perform repetitive tasks or control one application from another.

Microsoft provides COM interfaces for many Windows application services such as Microsoft Message Queuing (MSMQ) Microsoft Active Directory (AD) and Windows Management and Instrumentation (WMI).

Microsoft recommends that developers use the .NET Framework rather than COM for new development.

Microsoft Distributed COM (DCOM) extends the Component Object Model (COM) to support communication among objects on different computers—on a LAN, a WAN, or even the Internet. With DCOM, application can be distributed at locations that make the most sense to customer and to the application.

Because DCOM is a seamless evolution of COM, the world's leading component technology, developers can take advantage of existing investment in COM-based

applications, components, tools, and knowledge to move into the world of standards-based distributed computing. DCOM handles low-level details of network protocols so developers can focus on the real business: providing great solutions to the customers.

This component object models permit the creation of composable systems. They provide e.g., a binary standard for function calling between components, possibly bridging network boundaries. Furthermore they have a provision to group functions into interfaces, a way to dynamically discover the interfaces of a component, and a mechanism to uniquely identify components and their interfaces.

2.6. Component Based Groupware Study

Component based software has been chosen by some CSCW groupware application to support tailorability. It is because basically, component can be added at runtime. The following subchapter reviews 3 groupware systems which implement component based software technology.

2.6.1. EVOLVE (1998) [20]

EVOLVE is developed using JavaBeans component model to implement the search tool components and a modified JAVA BEANBOX DE as a platform for the search tool application. The modifications of the DE were required to permit run-tie tailorability of the application without having to dynamically generate and compile Java code.

Tailoring emerged as a cooperative activity, mainly with two of the users asking the h d, more technically inclined user to tailor special purpose search tools for them. During the experiment, search tool configurations (compositions) were automatically shared via a folder within the file system. The shared folder was accessible for all users and all configurations stored there were shown in a menu of the BEANBOX environment.

The JAVASOFT BEANBOX could only maintain the component structure of one local application during run-tie: It could not provide true distributed run-time tailorability, e.g. with the power user remotely changing the configuration of the running search tool on the machine of another user.. With the simple shared folder mechanisms for distributing configurations, a user in principle had to shut down the old search tool, wait for the changes to be made and then reload the newly tailored search tool fi-om the folder.

The system could not support truly distributed applications. It is because JavaBeans component model which does not easily support remote interaction in distributed applications.

2.6.2. DreamObjects (2003) [16]

The described DreamObjects platform offers services for data distribution, data consistency, concurrency control, and data persistency. DreamObjects is implemented in Java.

It provides developers with a methodology on how to develop collaborative applications and how to transform an existing single-user application into a collaborative one.

The transparency is achieved by the class hierarchy for the shared data objects. Its basic idea is to replace a developer-defined data object at runtime with a substitute. After an initial configuration, a developer can use the substitute like a local data object. The substitute hides the distributed actions that are necessary to keep a shared data object consistent. The class hierarchy does not restrict a developer, when implementing shared data objects. A developer can reuse existing data classes by inheritance, as he has not to extend any platform-provided classes. Additionally, he can feel free to compose data objects.

Application schemes for single-user as well as multi-user applications postulate a clear separation of user interface objects and data objects. DreamObjects supports this separation and offers a flexible notification service. A developer can use this

service to couple the user interface with the shared state. To be informed, whenever a new data object is created or an existing one is removed, a developer can define and register object-listeners. To track the modifications of a shared data object, a developer can define and register call-listeners. Apart from the normal callback mechanism, DreamObjects offers a mapping-based notification service. With the mapping-based service a developer exactly can define the information he is interested in and so avoids unnecessary events.

DreamObjects offers an object-based concurrency control scheme and a method based. The offered schemes can be applied per shared object. With the object-based one, a developer can implement floor control. Due to the nature of floor control, a developer explicitly has to handle the requesting and releasing of the floor. The method-based concurrency control scheme allows a developer to achieve a maximum of concurrency. A developer can define sets of methods that must be executed mutually exclusive. As DreamObjects supports composite data objects, there can be inter-object dependencies and these sets are not limited to one data object.

DreamObjects does not only support the common asymmetric and replicated distribution scheme, it also supports two adaptive distribution schemes. One of these distribution schemes adapts the distribution of a shared data object in relation to a user's working style.

DreamObjects supports two mechanisms to accommodate latecomers. One replays how the current state was reached. As not every latecomer is interested in a replay, DreamObjects also supports a direct state transfer. In both cases the latecomer support is fault-tolerant, e.g. the site that supports a latecomer can leave the session. It does not block the other participants in their work and works completely decentralized, i.e. there is no well-known site that supports all latecomers. As the latecomer support is completely integrated in the runtime environment, DreamObjects completely relieves a developer from the task to support the latecomer with a consistent state.

DreamObjects is based on a completely decentralized architecture, which avoids performance bottlenecks and makes the system much more reliable.

2.6.3. MACS (2000) [27]

The aim of this work was to develop a portable, flexible and scalable framework for CSCW applications allowing an easy, efficient and fast creation of customized components and thus the creation of a usage scenario specific system.

Portability is a key factor for easy deployment in a heterogeneous environment. The development of MACS used Java 2 as an implementation platform. The overall portability has already been a key factor during development, as MACS was implemented in a heterogeneous environment with different members of the development team working on Solaris, Linux and Windows based systems. The only real problem area hereby is the hardware access as needed for audio/video capture and playback. This problem is in parts solved by JMF, which at the currently available stage of development still exhibits a number of limitations.

Regarding the second key factor, flexibility, the MACS framework excels. Very different applications can easily be integrated into the framework and internal modules allow to extend/supplement the capabilities of the framework. This highly modular concept works well in practice and the configuration via the provided configuration editor is easy to handle. The two level setup of system and user specific configurations allows efficient system adaptation for different deployment environments. Despite its flexibility MACS manages to maintain a tight integration of its components.

The point of scalability of the framework has to be considered in two steps. First the scalability of the control and second of the provided application modules. The control is fairly scalable more so than, for example, ITU based systems. As demonstrated by the measurements even large lecture/class room scenarios can be efficiently supported. Further extensions to even larger scenarios are possible, but may require implementation of some of the marked enhancements, as again summarized in the outlook.

A further issue regarding the scalability of sessions is the availability and support by a suitable session controller module (especially regarding visualization. The same applies to scalability issues for reliable network protocols supporting group communication.

A further point of interest was the issue of interworking with other CSCW systems. Here the two introduced interworking modules show that it is quite well possible to provide such functionality within the MACS framework in a very flexible way. While interworking has been successfully demonstrated the limitations during interworking mainly caused by using a common (minimal) denominator have as well been highlighted. These limitations have to be considered for interworking scenarios. Here the advantages of interworking versus other

CHAPTER 3
ARGUMENTATION

This chapter describes the problem specifically, its possible condition, and some scenario to encounter the problem. In this chapter, the problem will be divided into several parts to ease the design.

The synchronous groupware system has to provide flexible adaptation and tailor-made configuration in the group collaboration.

Tailor-made configuration is including the selection of components and tools and their configuration according to special group parameters (e.g. number of members and their possible roles, openness of group) and scenarios (e.g. brainstorming, design session, text reviewing).

Flexible adaptation is regarding the change of situations during a session (e.g. allowing the creation of sub-groups) and behaviors of members (e.g. change of floor control policy is due to multicultural background of members). The system should be flexible enough to provide alternatives or to suggest alternatives according to group parameters or profiles of participants.

3.1. Requirement Analysis

The future enterprise groupware applications need to be able to cover the whole range of all groupware aspects in a way that is flexible and that can dynamically change [24]. Or it can be broadly summarized in two categories: (a) increased flexibility of groupware applications and (b) increased support of dynamism by groupware technology [24]. To meet this requirement of enterprise groupware, we will develop a highly flexible and tailorable framework for groupware.

3.1.1. Communication Strategy

The system is intended to provide synchronous communication strategy. Because the main aim of this system is to provide a synchronous collaboration work, same time different place. This is translated into the use of available different tools and application at the same time for the users who joined the same session. The variety of tools and application will be described later. But it dos not mean that the system will not use asynchronous communication, because asynchronous communication will undeniably be needed to do the management issues.

R1. The system should bring a synchronous system strategy

R2. The system will be based on client server architecture. The idea of client server architecture is by utilizing proxies in both sides, server and client, so that the invocations of program can be done as if the program is local.

R3. Communications between clients are based on client server architecture

Component based software is suitable to support software that has a high demand of tailorability because component can be added modified or deleted at runtime. Thus, we will design component based software. The communications of the groupware is basically communications between its components.

R4. To support flexibility and tailor-ability, the system is build per object component based.

Because the system is based on client system architecture, the software should be separated between master and client.

R5. The program must be separated between master and client (GUI)

The components will be separated based on its functions.

R6. The system must be separated between program, data objects and user interface objects components

3.1.2. Coordination

Coordination will be defined in some different way of work flow. The system will differentiate applications based on their type. Either it will be highly active or more to single person work and the others are just passive. The application will also be classified into the resources needed and the interactivity.

We will classify the applications based on the type of work that the applications can support.

R7. System will provide a list of applications for each phases of group work

R8. A group can choose from the list of applications, which one they want to use in the working session at runtime

To provide a continuous coordination, a database to save the state of the work is undeniably needed. The database will be hold by the main server. But the state of running history will be stored in each replicated master.

R9. Main server will hold a database of every session

R10. The database will save automatically every 15 minutes

R11. Before a session start, a group may continue working on the previous work stored in the database

R12. At the end of the session, user is asked whether he/she wants to save the work locally.

In this system, authoring mechanism will be translated into the use of different colors for each user. At session start up, the user is asked to choose 1 of at maximum 7 available colors, while the moderator will always have the color black automatically.

Every person will be differentiated by the color he/she choose. Different color is implemented as the color in the user profile box, and in some applications; e.g.

Brainstorming, Tele Pointer, Text Editor, Chalk Board. Therefore, it will be easier to notice who writes/draws what in the artifact.

R13. Authoring is implemented by using different colors for each member in the session.

3.1.3. Distribution

This system is aimed to give a highly distribution ability of the users. This groupware is designed to give its service to spatially distributed (worldwide) users.

According to Juzunovic and Dewan [13], it is found that:

- Under realistic conditions, a small number of users, high intra-cluster network delays, and large output processing and transmission costs suitable for the replicated architecture.
- While for large input size, the centralized architecture gives a better performance.
- While high inter-cluster network delays is best uses hybrid architecture.
- But high input processing and transmission costs, low think times, asymmetric processing powers, and logic-intensive applications favor both the centralized and hybrid architectures.

According to the implementation of PASSENGER [26], the connection between two continents, Asia and Europe, is not excellent. Learning from that experience, this system should be design to mitigate network problem that might lead to system's performance reduction. Under this framework, the system is desired to offer many different type of application; light, medium and heavy application. To support flexibility, the system will allow a variable number of users in a session, thus a suitable architecture upon this condition should also be considered.

Hybrid architecture is shown to provide the best design tradeoffs [21]. Thus in or system the distribution architecture mainly will implement hybrid architecture.

Only if all users join a session is located in the near area, the system implements centralized architecture.

R14. The system can flexibly operate in centralized or hybrid/semi- replicated architecture

R15. In a hybrid/semi-replicated, partitioning in the network is needed

R16. In 1 partition, there will be 1 replication.

During start up system will calculate cost of all variables concerning network latency between server to client as well as client to client intra and inter region, and processing power of each client. This information added with the information of number of users and type of application used in a session, will determined type of distribution architecture of the program.

R17. The system should be able to know the condition of the network connection to and between its client

R18. The system should be able to know the condition of resource power of each user

R19. The system is provided with the ability to give suggestion on distribution architecture as well as optimal partitioning based on user network condition and profile

R20. To give tailor-ability feature, the system will only give suggestion, while the decision is depend on users.

Because the system will have a possibility to implement distributed system strategy, in the case of hybrid/semi-replicated architecture, some basic problem of distributed system should be considered. Distributed system should provide some kinds of transparency; access, location, replication and fragmentation.

R21.The system must be provided with all basic distribution transparency; access transparency, location transparency, replication transparency and fragmentation transparency

Simultaneous input of distributed system leads to a problem of data consistency.

R22.The system must be provided with data consistency control mechanism; vector clock will be used to synchronize a shared object

Concurrency control should also be designed to mitigate the possibility of conflict of from users.

R23.The system must be provided with concurrency control, which will be translated into locking mechanism

Data replication leads to the problem of updating and synchronizing a shared state.

R24.Update and synchronization protocol is needed

3.1.4. Scalability

The system will serve a variable number of sessions. The users in a session may varies, according to the needs of the group owning the session. The system serves a large scale of users worldwide. User should associate to at least a group. A group should have profile regarding its organization, expertise, and purpose of using the groupware. A group of users should register itself to the system before asking a session it wishes to have.

R25.Every user must register to the system by giving the required profile

R26.Every group must register to the system by giving the required profile

R27.User and Group registration must be done before requesting a session

R28.A user can be associated to one or more group

Using computer mediation, in the brainstorming phase, a bigger group size can bring better ideas than the small one [4] [7]. In those experiments, small size group is represented by 6 or less people, while large size group is represented by 6-12 people [4] [7].

But in a more difficult task, that requires a more skilful knowledge, a group size of 3 people brings a better outcome than the bigger one [15].

Therefore, according to those researches, a maximum of users in a session should be different between applications. But if different applications restrict different number of users in a session, it will lead to a problem of regrouping during session. That is not what we are looking for.

For that reason, in this system, we will restrict the number of users in a session range between 2 and 8 people.

R29.The number of users may range from 2 to 8 people

3.1.5. Openness

Basically, this system provides openness to its users. It means that everyone can use its services. Every user who wish to use the advantages of this groupware system, may register for a session, ask for the slot time, and he/she will be able to use all functionality the system has. There is no restriction from which country he/she comes, from which organization, or for what usage.

But the openness of a running session is determined by the session owner—who register the session to the system before the session starts. The openness of a session can be translated to:

- Whether the system can be seen at the system activity view. It means that the system can either be seen by other people or invisible/anonymous
- Whether the system allow for late joiners. Allowing late joiners means that when a session already runs, there is a possibility to other people to come

late to the session by either invited by the moderator, or asking permission to the moderator to join the session.

Therefore, if a session status is invisible, there is no way that a late joiner may ask permission to join a session. But in invisible mode, a moderator is able to invite a user from his group to join the session late.

R30.System will allow session owner to define the session openness at session registration

R31.A group will provide the characteristics of the session; openness (allowing latecomers or not), role of each user in a group (moderator, observer, custom), floor control model, which phase is the session

3.1.6. Web exploitation

This system will exploit the advantages of World Wide Web to support the asynchronous part of the system. In the website, it will be shown:

- Who: name of group, name of member, the owner of the session
- What: the session name
- Where: the session member location
- When: the time slot and schedule of the session
- Why: type of session work phases
- How: the session characteristic

To provide security to the user, the system will allow session owner to set the data of the session, and the group as only a code, or as invisible.

R32.The website should provide 5W&1H information of sessions and users

R33.The system has capacity of 10 session at a time

R34. The system has an open table showing a free slot of time that can be chosen by a group. Each slot of time is defined as 1 hour

R35. A group may register (request) a session before the requested date and time of the session or directly before the session start as long as there are free slots

R36. A group can ask a session for one or more slot of time

3.2. Overview of the software

The basic demand to this groupware is to bring flexibility and tailor-ability feature. In a short description, based on the criteria made in [18][1], this system can be summarized as a groupware system that supports:

1. Functional Criteria, which specify what a user can expect of the system regardless of its environmental or non-functional constraints:
   - Messaging service
     - Synchronous: instant messaging/chat application
     - Asynchronous: offline notification regarding session management
   - Conferencing and Electronic Meeting Systems (EMS) → supported through video and audio conferencing
   - Group Decision Support → supported through polling application

[1] Please refer to "*Survey and comparison of CSCW Groupware applications*" [18] for each criteria explanation

- Document Management → supported through the usage of database for group's documents
- Document Collaboration → supported by a history during the session runs and authoring in artifact

2. Architectural Criteria, which define where and how the collaboration is managed:
   - Central Architecture (as a choice)
   - Hybrid Architecture (as a choice)
3. Focus Criteria, which define the focal point of collaboration:
   - Artifact centered
4. Time Criteria, which define the restrictions placed on the time of collaboration:
   - Synchronized Collaboration
5. Platform Criteria, which define the execution platform:
   - Platform independent (Multi-platform) Collaboration
6. User involvement Criteria, which defines the level of involvement required from the user:
   - High, as the user is forced to work with a different interface that he is used to in order to access the collaboration functionalities.

3.2.1. System Menu

This list of items will appear in the website of the system. It will be seen in the webpage that can be accessed for the first time by users who wants to use the groupware. This main menu will be placed in the left side of webpage. Every one including guest can see this menu. The illustration can be seen on Figure 5.

- About

- Registration
  - User registration:
    - Username
    - Password
    - Email
    - Location
    - Function/Expertise
  - Group registration
    - Group name
    - Group owner (username)
    - Organization
  - Session registration
    - Group name
    - Session name
    - Time
    - Duration
- Session Monitor
  - Session table; with properties: session name, starting time, duration, owner, number of member. If the session is in invisible mode, then no information is written but an occupied time will appear.
  - Join: to join a running session (only to a session which belong to the group of which the user is a member)
- **Download Client:** a freeware client program (User Interface part) is downloadable after User and Group registration
- **How To:** Manual of the program, FAQ
- Contact Us
- **Login** (if not yet Login)
  - Username
  - Password
- **Logout** (after Login)

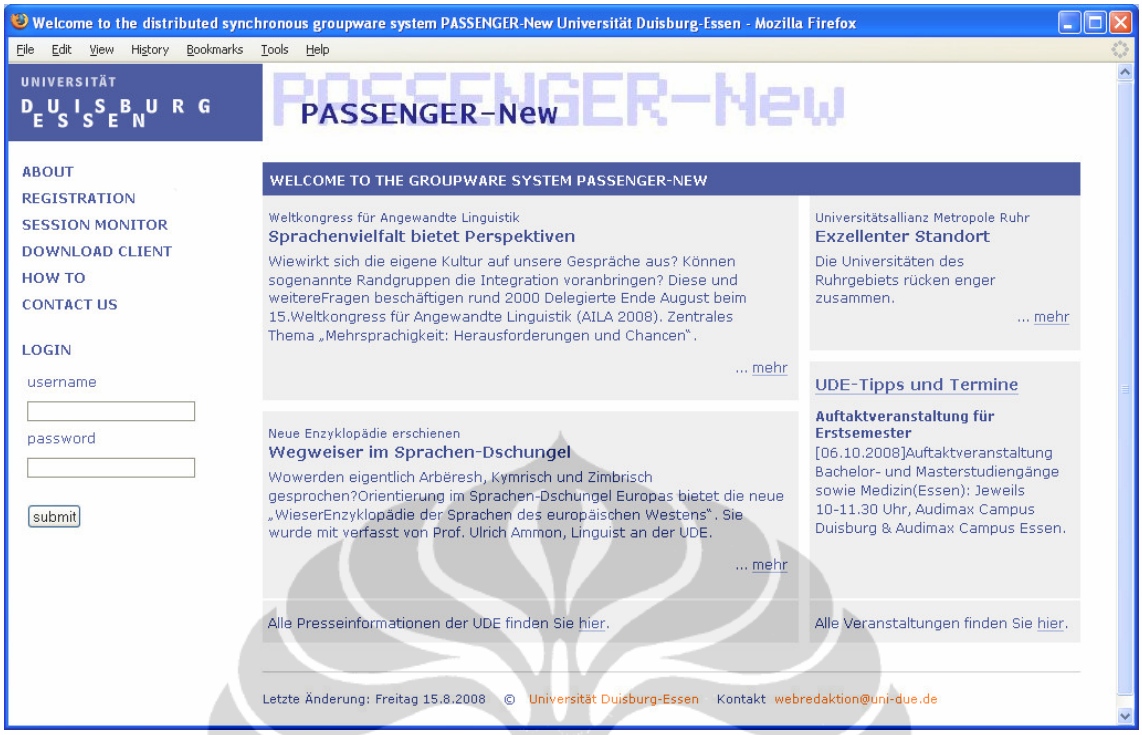

Figure 5. Main Menu Interface on Website

After a user login successfully, another menu will appear, placed on top right of the webpage.

- My Profile
  - Name
  - Password
  - Email
  - Location
  - Function
  - Photo
- My Group
  - List of groups:
    - Group name
    - Organization
    - List of member
    - Add member (special menu for group owner)
    - Confirm as member (special menu for group owner)
    - Delete user (special menu for group owner)
  - Join group
    - Search group name

- Join button
-

- My Session
  - My Session Table
    - Session name
    - Session owner
    - Time begin
    - Duration
    - Join Button
    - Invitation
  - Owned session
    - Session name
    - Edit mode (invisible or not)
    - Edit time
    - Edit duration
    - Invite user
  - Message

Figure 6 illustrates the user interface on webpage after successfully login. The default view is on My Profile.

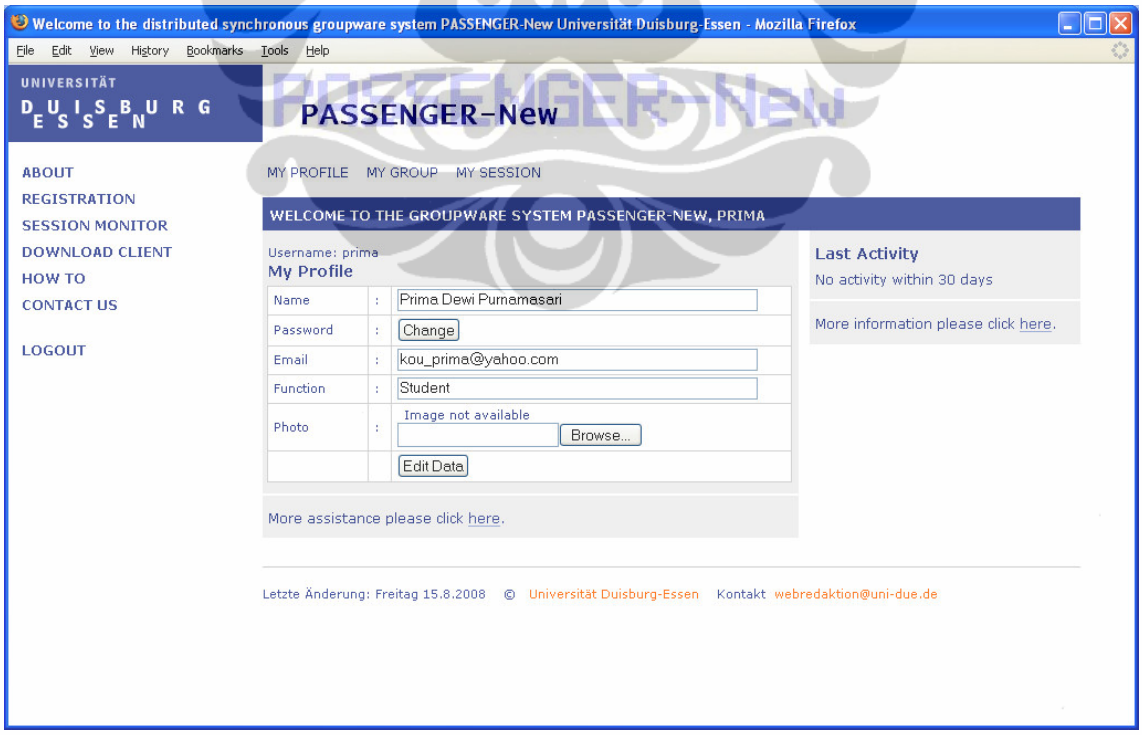

Figure 6. User Interface after Login

In My Session Table, which can be seen in Figure 7, each record represents a session, which will have different color:

- Grey represents an ended session. No join button
- Red represents an ongoing session which not allows late joiners or a session which already reach a maximum 8 people. No join button
- Green represents an ongoing session which allows late joiner. Join button available. When a person click join button, session code will appear which he/she will need to fill in the groupware to join this session. If he is already invited, a session code will appear.
- White represents future date session. Join button available. When a person click join button, a message will be sent to the owner. If he is already invited, a session code will appear.

Group Name	Session Name	Start Time	End Time	Duration	Owner	Member(s)	Code	Invitation
CSCW	Collaboration	17.8.08 9:00	17.8.08 10:15	1:15	MsYellow	5		
CSCW	Collaboration	20.8.08 9:00	20.8.08 10:45	1:45	Prima	8		
UDE	ABC	20.8.08 9:30	20.8.08 11:30	2:00	MrRed	5	sXr4	
Computer	Design	20.8.08 9:00	21.8.08 11:00	2:00	MrBlack	3	Join	
Computer	Presentation	21.8.08 13:15	21.8.08 14:30	1:15	MsBlue	4	Join	
UDE	DEF	30.8.08 10:30	30.8.08 13:30	3:00	MrBlack	7	t9k7	

Figure 7. My Session Table

3.2.2. Session Menu

This is the list of menu which will be appeared in the groupware in a session. This menu will appear on the top row on the groupware GUI. Illustration can be seen in Figure 8.

- Application Tools
  - Phase
    - Brainstorming
    - Presentation
    - Collaboration
    - Decision

- **Application** based on phase, an application may associate to more than 1 phase. It is a dropdown menu under Phase menu
    - Instant Messaging/Chat
    - Video conferencing
    - Audio conferencing
    - Presentation display
    - Text Editor
    - CAD Tools
    - Polling
    - Electronic Brainstorming (EBS)
    - Chalk Board
    - Tele-pointer

- (Session) **Tools**
    - **Leave:** Leave the session for individual participant
    - **End:** End the whole session, menu only available for moderator of the session to end a session before the time is up
    - **Invite:** menu only available for moderator of the session to invite new user to join the session

Additions to Session Menu, there are prompt boxes that will appear:

- Messages from late joiner that wants to attend the session. This prompt box will only appear to the Moderator. Moderator has choices either to Grant or Reject.
- Message from the system to every user whether he/she wants to save the work at the end of the session.

3.2.3. Workspace Functionality

Workspace functionalities use the biggest portion of display. It is below the session menu. User functionality will be located at the left side, while the Public workspace located in the right side. Unlike PASSENGER, in this system we do not implement private workspace in addition to the public workspace. It is because the use of private workspace often makes the data between public and private not update. And worse, passive user may lost his awareness of the current state in the public window, since he is busy working on his private window.

Unlike PASSENGER, this system will only use video display for the floor holder, while the passive members only shown by his/her name and properties (function, location).

Below is the list of workspace functionality menu:

- User functionality
  - Floor Holder display box, with name, function, location
  - Lists of participants, with name, function, location
  - Ask Floor
  - Interrupt, enabled only for Member
  - Ask to observe, enabled only for Member
  - Interrupt permission, enabled only for Floor Holder
  - Invite
  - Instant Messaging/Chat, as an application that always available, but if users do not want to use it, it can be hidden
- Public workspace
  - Working space
  - Tools, depends on application
  - Commit button
  - Abort button
  - History
- Application per phase:
  - Brainstorming
    - Video conferencing
    - Audio conferencing
    - Electronic Brainstorming (EBS)
    - Chalk Board
  - Presentation
    - Video conferencing
    - Audio conferencing
    - Presentation display
    - Tele-pointer
  - Collaboration

  - Video conferencing
  - Audio conferencing
  - Text Editor
  - CAD Tools
  - Chalk Board
  - Tele-pointer

- Decision
  - Video conferencing
  - Audio conferencing
  - Presentation display
  - Polling
  - Chalk Board

In Figure 8, it can be seen the illustration of the user interface in a session. It is the main interface of the synchronous groupware. At the top level, Menu of the session, Phase (with dropdown menu Application), Floor Type, Tools and Database are present. At the left side there is video of the Floor Holder together with his/her identity. Below Floor Holder Identity there are list of user identity that join the session. Each user will have different color represent with little box with color nest to his/her name. Below it, there is Chat Box that is enabled all the time. The biggest portion of the user interface is the working space at center point. On the right side, there are tools correspond to the Application used in the session, Logger (both HL and TL) with the tools Commit, Abort, Undo, Redo (using icons). At the status bar at the bottom of the interface, user can see what the actual activity of the session is.

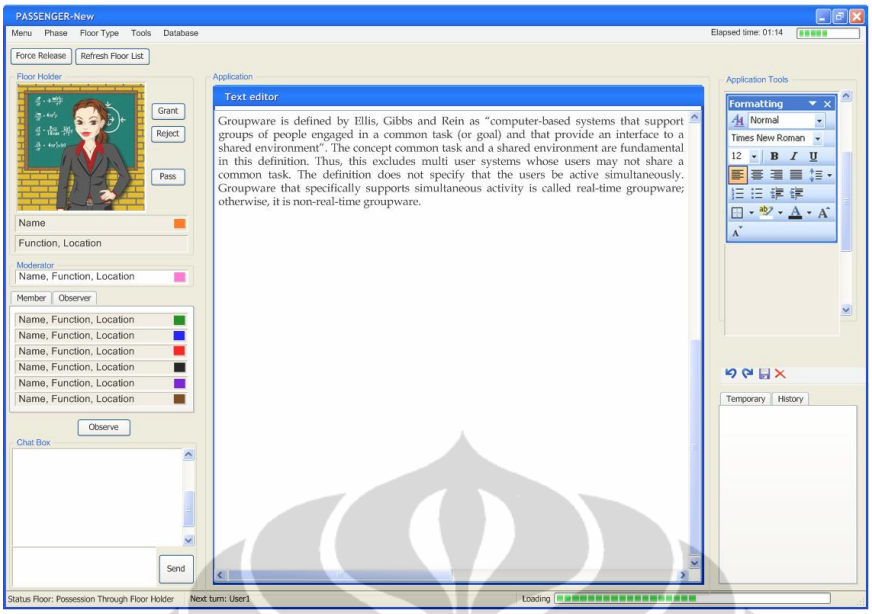

Figure 8. Session GUI