NTNU
Norwegian University of
Science and Technology

Collaboration Instance Manager of UbiCollab 2008

Collaboration Instance Synchronization and Management in P2P network

Xiaobo Wang

Master of Science in Computer Science
Submission date: June 2008
Supervisor: Babak Farshchian, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

The problems I need to resolve in my research and describe in this report include how to develop a pure Peer-to-Peer network for UbiCollab and Collaboration Instance Manager. How to make data synchronized between peer and peer. How to manage these synchronized data in a persistent, logical structure. As a ubiquitous groupware system, I also give the approach of how to use location information to update user's information. Finally, for show the result of my work, I need to develop a Graphic User Interface (GUI) to show the utility of this system.

Assignment given: 15. January 2008
Supervisor: Babak Farshchian, IDI

Abstract

This report is for my research of Collaboration Instance Manager of UbiCollab project. UbiCollab want to be the platform for ubiquitous collaborative active. UbiCollab project aims to develop a distributed collaborative platform which makes people in distributed space ubiquitous collaborate with friends and colleagues. Collaboration instance manager (CIM) is a core component of the UbiCollab platform, which manage such collaborative activities.

My research topics of CIM include in the P2P network development by using JXME, the data synchronization through this P2P network and how to manage these synchronized date by using a local file system. The result of my research is a CIM system, which deployed as OSGI bundle. User can use that do some collaborative active. This CIM system manage the service level of data synchronization, other modules and applications can use that to handle data synchronization between each other without know the details of how to implement it.

For that purpose I first reviewed the related theories of distributed systems, ubiquitous systems, mobile systems and CSCW. After that review I researched on some alternatives for developing such system and choose the candidate technologies for my prototype. Secondly I analyzed the requirements of UbiCollab and designed the prototype. Based on that design, I implemented and tested that CIM system based on agreed common scenarios and developed a simple GUI for show the utility. Finally, I evaluate the system by analysis system requirements and scenario criteria.

Table of content

Chapter 1 Introduction 1
  1.1 Overview of UbiCollab project 1
  1.1 Motivation 2
  1.2 Problem define 3
  1.3 Research approach 4
Chapter 2 Theoretical review 6
  2.1 CSCW and groupware 6
  2.2 Data synchronization and concurrency control in real time groupware system 7
  2.3 Mobility and location awareness 8
  2.4 Ubiquitous system 9
  2.5 P2P network and JXTA 9
Chapter 3 State of the art 11
3.1 Introduction 11
  3.2 Plan B, how file system used in ubiquitous system 11
  3.3 JXME, used for develop a P2P network on mobile device 12
    3.4.1 Content management system 13
    3.4.2 Myjxta 13
    3.4.3 Gnutella 14
  3.5 Collaborative editing system 15
Chapter 4 Requirement specification 16
4.1 Introduction 16
  4.2 Quality requirements 16
  4.3 Functional requirements 17
  4.4 Non-functional requirement 17
Chapter 5 System Architecture 20
  5.1 Introduction 20
  5.2 System logical static view 20
  5.3 Collaboration Instance 21
  5.4 CIM and JXTA services 22
  5.5 Process view and MVC pattern of CIM 22
  5.6 CIM classes 25
  5.7 System overview 26
Chapter 6 Implementation 27
  6.1 Introduction 27
  6.2 Scenario for demonstration 27
  6.3 Test bed 29
  6.4 From JXTA to JXME. History and change issues of the P2P network development 31
  6.5 File system for UbiCollab 32
  6.6 Synchronization of file system 33
  6.7 Space proxy and location awareness 34

6.8 CIMTool 34
Chapter 7 Evaluation 39
Chapter 8 Future work 42
Chapter 9 Conclusion 43
Appendix 44
A 1 Common scenario 44
A 2 Code 55
A 3 Deployment 98
Reference 99

Chapter 1 Introduction

1.1 Overview of UbiCollab project

UbiCollab is the platform for supporting collaboration through network.[1] As the result of internet capability and technical evaluation, communication and complex collaboration among physically distributed people become possible. Many academic communities research on CSCW and groupware, which closed related to distributed collaboration. Other interesting aspects for collaboration are mobility and ubiquity, which been considered as high logical level of collaboration and treated as inherent properties of UbiCollab project.[1]

UbiCollab is the intersection of these areas mentioned above since it takes all of these researches filed into consider. It supports the groupware liked common collaborated services and makes this kind of collaboration to be ubiquitous. The mobility makes the ubiquitous collaboration becoming possible, since UbiCollab should be deployed on both PC and PDA or cell phone.

I mention some important concepts of UbiCollab here for easy understanding of this project and give the details of them in later chapters. *Collaboration Instance* (CI in short), is the virtual collaboration space (context) where services, resources to be shared and collaborated works happen. Physical distributed users use UbiNode, which is the device deployed by UbiCollab system, to collaborate with each other. These users¡ services and resources in physical space are published to CI, a common shared space. Through this way, people, services and spaces construct the so-called *human grid*.[1]

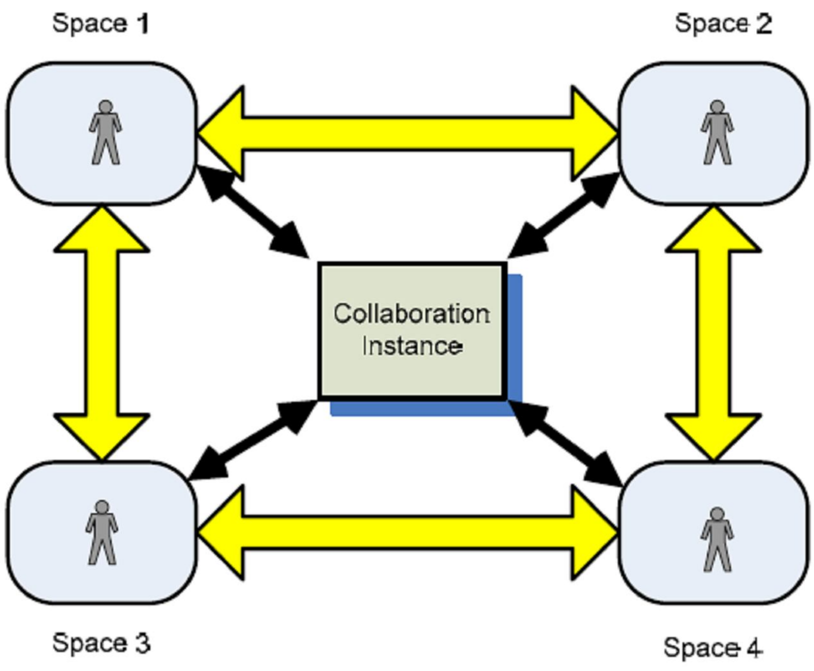


Figure 1.1 an instance of human grid

Figure 1.1 shows an instance of human grid. The Space represents physical space, CI in the central of grid. One important property of UbiCollab human grid is: the infrastructure could be re-constructed, which means the topology of grid network and participants might change due to spontaneous leave, join and location change of users. This character is the requirement of ubiquitous computing and CSCW. I will explain them later in literature review chapter.

UbiCollab project is been divided into some sub-tasks of student projects based on different modules. The modules been research on in this semester are: Session manager, Collaboration Instance Manager, ID manager and Service Discovery manager. I worked on Collaboration Instance Manager (CIM in short), and cooperated with others who responsible for the three modules.

UbiCollab is an open source project, which means the developed source code is free for use and the libraries, platforms used for development must has open source license. The wiki homepage of UbiCollab is: http://mediawiki.idi.ntnu.no/wiki/ubicollab/index.php/Main_Page.

1.1 Motivation

As mentioned above, UbiCollab means Ubiquitous Collaboration system. It is not just a simple combination of ubiquitous computing and CSCW, groupware. We want to develop a platform for people to cooperate with friends and work mates in the natural way. Ubiquitous environment and mobility is the key for success.

There is a lot of research on CSCW from 1984.[2] In early time, researchers and the system developers mostly without consider how people collaborate with people in natural way. The reason is both due to technical constraints and no theories supporting. These systems and applications, of course, support some level collaborated activities to users who are locate in different physical spaces, but at the same time, undermining the mobility of users , which is what they used to in natural way. [3]

Mobility is closed relate to ubiquitous computing (ubicomp) since the later one means we can access information and get service anytime and anyplace. [4] For this purpose, not only mobile devices such as PDA and cell phone but also ubiquitous environment are required.

UbiCollab project is supposed to be the platform through which people can do ubiquitous collaboration. We deploy system on PC, laptop and mobile device. User can take these mobile devices to different physical spaces, which represent different collaboration context. UbiCollab supports collaboration, context awareness, spontaneous participant and automatic re-construction of human grid. The achievement of last one based on location or other context parameters change.

My contribution to UbiCollab is CIM module. CIM is the basic core of UbiCollab system. I list and explain the main problems CIM and UbiCollab faced with in next section.

1.2 Problem define

I define the main problems of UbiCollab and my CIM module, both technical and non-technical ones, into two categories. The first one contains the lower level communication between peers, implementation of the infrastructure network. The other is in a higher level abstraction, relate to ubiquitous, context awareness and usability.

Following are belonged to the first category:

P2P network development

Because of the human grid and ubiquitous requirement of UbiCollab, spontaneous and no dependence on server is very important to the network. P2P is the best choice in our case. I responsible for such a P2P network development, which supports basic data transfer and complex data synchronization. I use JXTA [5] for this purpose, through which, each UbiNode becoming a JXTA peer and collaborate with others in the same group.

Data synchronization

One of the most important topics of distributed system is how to keep the data reside in remote peers consistent with each other. The ideal situation is all the peers have the same data and view of it. But it is impossible since network delay and currency operations collision. UbiCollab, as a groupware, in additional to the classic synchronization issues in distributed database, has its specific property. The feeling of human and performance must be take into consider.[6]

Data synchronization and currency control are very wide and deep research area. In this thesis report, I focus on how to use JXTA to achieve synchronization purpose, how these synchronized data and information are stored in a logical way and, how to use synchronized information and data to make UbiCollab context awareness. At the same time, I will give my idea of how UbiCollab should treat with currency control.

Deployment of device and platform

In order to fulfill mobility, UbiCollab have to be installed on mobile device such as PDA and cell phone. I have implemented an older version of CIM on standard PC and laptop. There are some constraints and technical problems when change that to CDC mobile devices. I will give a trace of CIM development and the problems encountered such as compatible and wireless network.

Following falls into the second category:

Context awareness

Context awareness is important to CSCW. This awareness contains knowledge of your cooperators and this knowledge facilitates collaboration activities. In UbiCollab, such awareness is location awareness. How to design CIM in order to make it receive and make use of the location information should be taken into consider. In human grid, when user changes the physical space, the location awareness information might be: ¡ I leave my office so I don¡t want talk about work!¡ or ¡ I¡m home now¡ . The information and space change could cause user leave and join a new CI, or just find new services and publish it to CI. CIM has to know how to treat with these raw data from sensor and do corresponding update based on it.

UbiBuddy

UbiBuddy is the demonstration application of UbiCollab. It is a MSN liked chatting, collaboration platform. My work mates and I want to use it show the basic functions and services UbiCollab can support to users. It is a combination of the modules we developed and its GUI affects how user feels of and uses UbiCollab.

Persistent storage and local data management

UbiCollab system and CIM should keep data persistent in UbiNode. The data include the service resource and other user information. This is not only for initial CIM and UbiCollab system but also used for data synchronization management. The storage structure decides the destination where synchronized data will be propagated to and local CIM system can retrieval resource from.

1.3 Research approach

Firstly, I read and research on papers and books relate to CSCW, data synchronization of distributed system, JXTA development and how to using file system to develop ubiquitous system. The literature review and study give me basic knowledge and technical supporting for my research work.

Beside the theories and technologies mentioned above, I do some case study focus on some counterpart applications and alternatives technologies. The state of the art research not only gives me some example of how groupware, ubiquitous systems should be developed but also what shortcomings should be avoid during my design work. After the analysis, we know what aspects make UbiCollab and CIM unique and different to others.

I have developed a CIM module based on JXTA standard version. For my thesis I develop a new one (not dependent on the older one). For that purpose, I compare what is different between JXSE and JXME. I list constraints and problems when I use the micro version and what can be improved in future.

What I mentioned above is preparation work in early phase. After that I design system architecture based on requirement specification analysis. A good understanding of both technical requirements and non-technical ones limit the boundary of CIM. This boundary decides what I should and what I should not do. The system architecture includes different views of CIM and how it communicates with other modules, proxies and applications.

I implement a CIM version 0.1 for demonstrate the usability based on the design work. My work mates and me decide to use UbiBuddy[1] to show what UbiCollab system should looks like and how it work. CIM supports CI manager and P2P network initial for that application. I describe the details of implementation in the chapter six.

Finally, I evaluate the system based on how it satisfies the requirement and performance. I also give a conclusion of how it can be improved in future in the end of this report.

Chapter 2 Theoretical review

2.1 CSCW and groupware

CSCW (Computer Supported Cooperative Work) should be conceived as an endeavor to understand the nature and requirements of cooperative work with the objective of designing computer-based technologies for cooperative work arrangement. This is the definition of CSCW given by Schmidt and Bannon (1992)[7]. In this definition, CSCW is basically a design oriented area, a design discipline but not a strict guideline or framework. Groupware is the implementation of CSCW. It supports the platform for collaborated work on. As the definition of UbiCollab project, it belongs to CSCW research area and the real implementation is a groupware.

Cooperative work is inherently distributed, this character result in the complexity of cooperative work. The more distributed the activities of a given cooperative work arrangement, the more complex the articulation of the activities of that arrangement is likely to be[7]. The CSCW system design should consider this characteristic into cooperative design.

In order to support distributed collaboration, a common shared space is required. CI is this abstract space in UbiCollab system. For the common information space, CSCW system should support basic structures for establishing and maintaining conceptual structures to keep track of actors work and negotiate of the update of work for a common consensus.[7] CIM is the core module responsible for these purposes, it should provide data publish, remotely request and access, event registry, notify of update and space management.

In the white paper of UbiCollab, authors mentioned the collaboration instance is a virtual context for collaboration,[1] combine with the awareness for cooperative work, we can say the UbiCollab should support contextual awareness. In order to understand Context-aware computing we must know what is context in this domain. Context is: any information that can be used to characterize the situation of entities that are considered relevant to the interaction between a user and an application, including the user and application themselves. Context is typically the location, identity and state of people, group, and computational and physical objects.[8] In UbiCollab systems, CI should contains and interacts with physical devices in the surround environment, the states changes of these devices gives contextual awareness of people in the environment, for example you turn on a light in you office reveals you are working now. CI also interact directly with the physical space around you, for example the locate track and location based services, awareness systems.[9] These physical changes will reflected in CI which eventually gives a contextual awareness to you friends or mates in remote places. I will give some details of location awareness in later section.

2.2 Data synchronization and concurrency control in real time groupware system

A distributed system is the one in which hardware or software components located at networked computers communication and coordinate their actions only by passing messages.[10] UbiCollab is a typical distributed system if taking this definition to consider it: different nodes (Ubinode) coordinate actions and sharing resource by messages passing on the connected network, which can both mobile and cable network.

Data synchronization is one of the basic services that distributed system should support. Synchronization achieved by data replication and concurrency control. The idea is, keep the data in distributed node consistent with each other.[10] Replication of data means maintenance of copies of data at multiple computers. In the system model of replication, copies of objects or data called *replicas* held by distinct *replica managers*, which contain the replicas on computers and perform operations upon them. In UbiCollab, replica is the data shared in CI, and CIM act as the manager of such data.

When we work in a physical shared space, we work on a distinct objects and to be physical constrained from doing particular actions.[6] However, situation becoming complex when we use groupware liked distributed collaboration system. These systems have multi copies over distributed computers, data replication only can replica data to them but not make them consistency. Concurrency control problem arise when concurrency operation happened on multi peers. Time delays when exchange such concurrency conflict actions.[6]

Traditionally, there are two ways for managing concurrency conflicts. One way is serialization and another one is privileged access through locking. Serialization usually achieved by generates a total ordering of events. A scheduler then decides how to execute events or how to detect and repair order inconsistencies.[6] Locking, in another way, keep privileged access to object. Only the locker holder can access and operate on the locked object and other candidates should request to lock. Commonly, there two policies as guideline when these approach are used: Optimistic and non-optimistic. Optimistic policy has the assumption that: there are very low opportunities for conflict operations and concurrency collisions happen. It allows data replication between peers without ordering guarantee before, if inconsistencies be detected, some rules used to repair that. Non-optimistic, in another way, make sure the arrived event must in the correct global order before allows it affects local state.

Groupware is the tool for help people collaboration, human interaction and feeling has to be taken into consider when design concurrency control approach. The result of such control must to be perceived and understandable to human. And the local response should be as soon as possible. The two traditional approaches mentioned above get problems when used in groupware. In non-optimistic policy, both approaches face the problem of waiting and pending either for global order guarantee or request lock. In optimistic policy, the requests and operations can be allowed immediately, but the problem due to how interface to show repair of inconsistencies caused by out of order sequences through undo and transformation.[6] People will feel strange when the co-edited object, such as text, change to an unexpected state due to transformation or rollback to the original state caused by redo. But worse situation is people not notice of such change, this always causes data lost or failure of collaboration.

As mentioned above, the choice of concurrency control for groupware is different to other distributed systems. We should consider both human and technical aspect. From user¡s point of view, the data update and repair should be understandable. When user editing characters or grabbing pictures, unexpected characters change or picture move should make them confused. At that time, a lock policy might be a good choice. On the other hand, if user want to work close with others and don¡t want to grab the control, a serialization policy may suffice.[6]

In this report I mainly focus on how to use UbiBuddy liked interface and CIM, JXME based p2p network to achieve data synchronization. The user information and local update should be synchronized to others on time and in an understandable way. I will give details of that in implementation chapter.

2.3 Mobility and location awareness

One important character of collaboration active is mobility.[3] In physical space, we not only stay at the front of desktop but also move to lab, office and go home. In CSCW and groupware, we want make use of this flexibility to give users better collaboration experience. A low level but obviously good example is mobile phone, which facilitate us everyday life and make an evaluation of communication technology.

Based on wireless network, cellular network and mobile device such as cell phone and PDA, we can get touch with friends and colleagues anytime and place. But is that the final result we want to? In CSCW, this is only the basic requirement for advanced, complexly collaborated work. Groupware should base on, but have more advanced functions and services than text, voice, video communication, such as co-edit and group meeting. On the other hand, connect to friends and colleagues anytime could break some social rule, for example, I don¡t want to talk about work when I home.

In groupware system design, we need to make use of mobile device to make cooperate work can be done during movement and consider the information from location change. This information is location awareness.[9] UbiCollab wants to develop the platform that satisfies such requirements. UbiCollab platform should be deployed on both static devices such as PC laptop (even laptop could be use during movement but not in and easy way) and mobile devices. The platform should collect information from environment in order to give location awareness to services and applications. These applications then update and adapt to the changes, in order to give users and co-works awareness. A scenario is: Charlie walks into office, his status of CI ¡work¡ becomes online, which means he is work now and available if the participants in the same group want to have some cooperate work with him.

The location information could be collect from multi ways, such as GPS. In our case, we use RFID tag fro that purpose. The RFID tag should contains location information and read that through a Bluetooth reader.

2.4 Ubiquitous system

Mobility makes groupware has one character of work in natural way, but real collaboration not only needs move flexibility but also contains more activities. We use artifacts in physical space to help us work, such as whiteboard and printer. The purpose of ubiquitous computing is: augment people ability of treat information and cooperation by support services which vanished into physical environment (disappear).[11]

In order to develop ubiquitous system, some themes need to take in to consider. One is *natural interface*, which make people use the natural way to interaction with computer systems, such as speech input. The purpose is to make people interact with computer like what he or she used to with physical world. It is contribute to disappear because when you interact with systems in a used way, you will not pay attention or at least less to the interaction itself but more focus on the task or goal underlying it. Another is the *context-aware*; I have mentioned some theory about context especially location awareness. In ubiquitous system, the consumers of context information are both human and the system, the system adapt itself to the environment it involved in based on the context it get, in this way system can support services to human base on different situation. Finally, the large number ubiquitous applications want the *capture* the life experiences and *access* it latter.

UbiCollab research mainly focus on how to get services from environment and support context awareness. These services then shared with co-workers, who can use them supported by context awareness.

2.5 P2P network and JXTA

I mentioned human grid before, this grid should support users with spontaneous join and leave. Another reason for spontaneous and independency is mobile devices are easily disconnect from network both due to low battery and wireless network unstable. For this purpose, UbiCollab needs Peer-to-Peer topology network. The advanced feature of P2P is: all the peers have the same role, no dependency between each other. This feature means that there no central server, which supports central control or storage for others in client-server architecture.[10]

Because of independency, peers could join and leave without affect others. However, no central control arise some problems. One of them is route, how peers discover interested peers or resource and then replicate dynamic date to them. Another one is shared space. CSCW, groupware must provide common space service, but common space naturally relate to central control and dependency.

I use JXTA to develop the P2P network used for UbiCollab and provide low level functions for

CIM. JXTA, which is a programming language and platform, designed to solve a number of problems in modern distributed computing, especially in the area broadly referred to as peer-to-peer computing, or peer-to-peer networking, or simply P2P.[5]

I use JXME, the micro version of JXTA to develop the foundational network and platform. One reason of use JXME is that: it easily solves the route problem by use pipe. Another one is, JXME proxy-less version is tested compatible to deploy on PDA, which shows mobility for UbiCollab. I will describe details of JXME in next chapter.

Chapter 3 State of the art

3.1 Introduction

There are a lot of research have been done in groupware, P2P network and resource sharing. I research on them and give a conclusion in this chapter. I compare these results with UbiCollab and CIM in order to show what ideas CIM and UbiCollab could get from them, what is different between them and what is the advanced of UbiCollab and CIM.

3.2 Plan B, how file system used in ubiquitous system

Ubiquitous system provides users access to different services, which vanish into physical space. These services might have heterogeneous runtime platforms, APIs and protocols. Developed ubiquitous system need makes these heterogeneous systems interoperate with each other. The ubiquitous system always has new interfaces, which incompatible with legacy applications and services. We need develop new tools for ubiquitous system even the old ones are work well.

Middleware generous used for integrate different systems. Middleware approaches rely on XML-based interfaces for interoperability. One reason for this is to provide a universal interface that most programs understand. However, because their interfaces are also new, general-purpose tools stop working for the new abstractions, so programmers need new tools even when old tools could work. In Plan B and Plan 9 projects, researchers use a hierarchy file system instead of XML tree to represent resource and services. This approach require only file explore system but doesn¡t need introduce new technology and software for both people and programs to use it.[12]

Resource located at corresponding file path, which represented by resource volume. Volume name identify resource exported in a global namespace. The operations on this file system are simple open, close, read and write. By using special requesting commands, programs and people can easily find, modify and request resource. User can also manually use third-part tools or applications change the file content, for example change the status of a light from **on** to **off.** The change will cause light turn off since system support file monitor.

I adopt some ideas from the file system approach into CIM and UbiCollab system. I define a folder hierarchy used to locate different resource and information, such as the services that users published into space, user status and common session templet. A XML file describes the concrete content of resource. CIM should responsible for manage these files, notify modification and synchronize them. The reason for use this approach is because it is easy for both human and system to understand and show usability. CIM create and manage the folder hierarchy by JAVA, which means it works on both Windows and Unix system. The details of architecture and implementation will be described in later chapter.

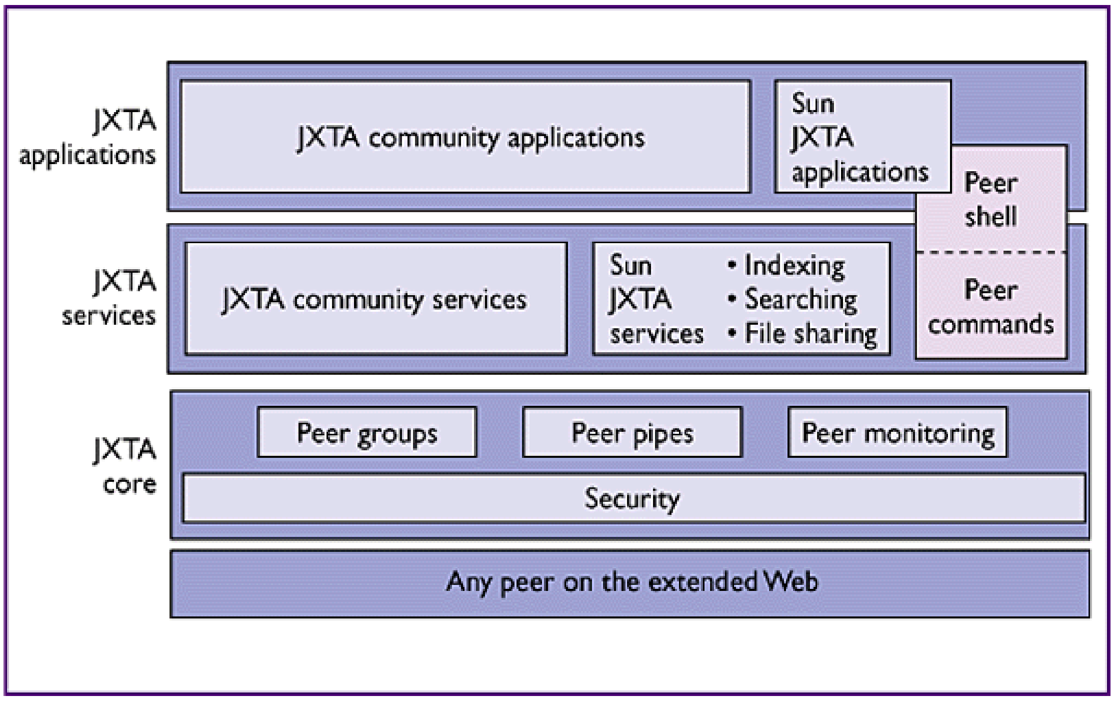
3.3 JXME, used for develop a P2P network on mobile device

JXTA is the platform used to develop P2P network, which responsible for network communication. The communication and data transfer through pipe, which is the channel that encapsulated message send to target peer(s).

The main objectives of JXTA are: Interoperability, Platform Independence and Ubiquity.[5] JXTA project provide protocols used as mechanism to realize the objectives. The basic component in JXTA world is peer, peer can creates, join, peer group and publish advertisement, then peers communicate by passing message from pipes or sockets.

The typical JXTA software architecture is illustrated in figure 3.2, which is divided into three layers. At the bottom, the core layer deals with peer establishment, communication management.[5] The service layer supports core services are the services all the peers in JXTA world must implement, additional to these, some standard and user customized ones are implemented. Different peer groups are recognized by the services implemented inside this group, in actually, group join and creation is the process of instantiate a group object with the predefined services. On the top of this architecture are applications, which use the services supported by lower layers.

Figure3.2 JXTA software architecture [5]



JXME is the micro version of JXTA, which used for develop system deployed on mobile devices. It has two distributions. One is CDC (proxy-less) version and another is CLDC. The first one used for my thesis research. The reason for choice that is both due to it is tested compatible with PDA with windows mobile system and it is similar with the original JXSE version. The main differences between JXME and JXSE are: JXME requires more support from rendezvous peer and not support cache, since the limitation of mobile device capacity.

However, JXME proxy-less version can work well without rendezvous peer in a local sub-network, which is the environment for my research. This feature gives possibility for develop a pure P2P topology network, no dependency and relay on others. In future work, the Ubi-network could be extended to whole internet with deployed rendezvous peers, who facilitate discovery, support communicate across NAT and firewall and provide utility services.

3.4 Shared resource management

Common space and shared content is required by UbiCollab project. I do research on this topic and give some possible alternatives for my thesis and future work.

3.4.1 Content management system

Content management system (CMS) is a desire to manage the explosion of content, a desire to provide structure and meaning to content in order to make it accessible, and a desire to work collaboratively to manipulate content in some meaningful way.[13]

However, traditional CMS systems face problems with the P2P network. The ranged capability of mobile device market requires the CMS should treat with a lot of heterogeneous or low capacity platforms and devices.

There are some projects addresses the problem of how CMS work in a P2P network. JXTA CMS is the extension library based on JXTA. By using that, developers can develop a shared content space and access them between each other. However, this library tested not compatible with JXME CDC when I tried to develop a JXME based CMS approach. In my CIM, I don¡t use CMS approach to develop the shared space, but it is might possible in future.

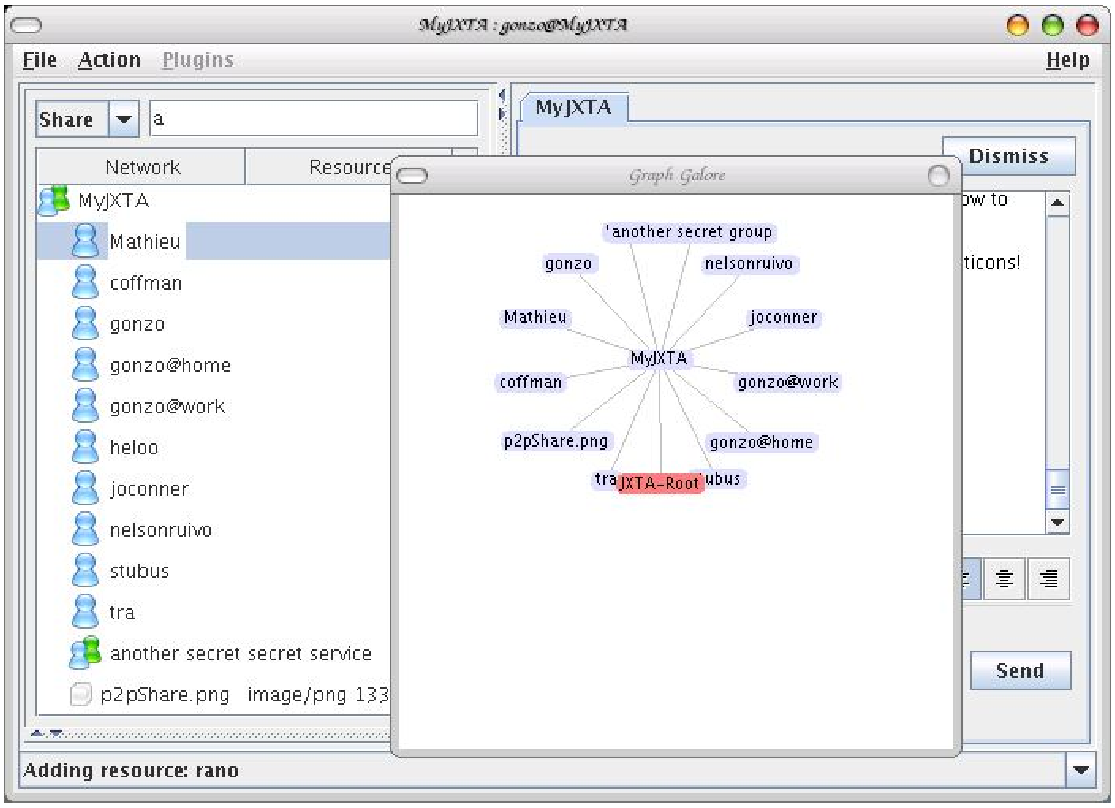
3.4.2 Myjxta

An open source project developed a JXTA based application: Myjxta, which show some usability of JXTA, such as peer discovery, group discovery and file sharing. CIM should support some services like the Myjxta did but more focus on CI management, data synchronization and awareness. In a short word, CIM use JXTA to achieve services implementation rather than a JXTA application.

Myjxta use JXTA CMS developed the file sharing system, by point at desired file through local file explore system, it create a shared file folder and other peers in the same group could access them. However, as I mentioned above, CMS is not compatible with JXME and UbiCollab require OSGI as the container. I didn¡t adapt Myjxta to CIM implementation both due to too much change work required and incompatible.

Figure 3.4.2 shows screenshot of Myjxta, you can see there are share function button and list of avatars represent discovered or join the same groups peers. The right part shows the relation ship of groups, all the custom groups created from the initial group: MyJxta. This approach used in UbiCollab design. At beginning, the initial peer (called UbiHome), create the initial group called UbiCollab. All the peers want to participant UbiCollab network should join this group.

Figure 3.4.2 Myjxta user interface



3.4.3 Gnutella

A successful P2P case is Gnutella. Gnutella is a file search and sharing system, like other P2P application, Gnutella builds at the application level a virtual network (Gnutella network) with its own routing mechanisms.[14] Peer in the Gnutella network query and propagates messages by using a flooding mechanism, which means query is propagated to all neighbors within a certain radius.[15]

We use a standard taxonomy in P2P world to classify different systems. Consider Degree of centralization, there are purely decentralized, partially centralized and hybrid decentralized systems. Consider the network structure there are structured and unstructured systems. Gnutella is a pure centralized, unstructured network system.[16]

In UbiCollab project, we want to develop a pure decentralized P2P network without any central server dependency. No dependency is more advanced than Gnutella liked centralized system but at the same time, hard to implement and cost more network resource and time for route. In future work, rendezvous peers could be taken consider into, since it support some ¡super service¡, which facilitate resource discovery, however, becoming system to be a partial P2P topology. I want to develop a pure P2P structure system, peers participant in which replicate available resource or the reference to them to interested peers.

3.5 Collaborative editing system

Collaborative editing is another important function groupware could support. Users could co-edit text, graph and some complex objects. The challenge to co-edit is to make data consistent in all sites. Basically, collaborative editing system should follow three criterions during deign and implementation. Intention preservation, casual consistency and convergence.[17] Intention preservation requires that for ¡any operation *op*, the effects of executing *op* at all sites are the same as the intention of *op*, and the effect of executing *op* does not change the effects of independent operations¡.[18] The operations on each site should execute on a state where they are legal. Such as an insert operation must relay on the two characters before and after the target one. The respect of preconditions ensures the causal consistency criteria. Convergence criterion states that peers with the same set of editing operations compute the same state of the replicated data. A direct way to ensure convergence is that the state of the data does not depend on the order that a peer executes received operations.[17]

In order to achieve these requirements, system should consider how to treat with arrived operations. Because of concurrency operation and network latency, arrived operations could out of order in different sites, which could cause different result if without concurrency control. As mentioned above, lock and serialization are the common technology used for that purpose. But for P2P and groupware system, more situations should be taken into consider as I described in last chapter.

In my report, this kind of co-editing system is not the main research area. Because of the time limit and complex of concurrency design. However, I will give my ideas of possible approaches of how co-editing application could be achieved through CIM and possible extension in future work.

Chapter 4 Requirement specification

4.1 Introduction

All the design and implementation work based on requirement analysis. I should understand the requirements from stakeholders in early phase and define boundary, which tell me what should do and what shouldn¡t. I list these requirements specification into three categories and give constraints of future implementations.

4.2 Quality requirements

These main quality requirements from a generally point of view to see what features CIM and UbiCollab should focus on. In real design and implementation, I mainly focus on CIM. However, since quality requirements are required by the whole UbiCollab system and affect CIM. They should get from functional and non-functional requirements. However, I list them firstly in order to give an overview of CIM and UBiCollab.

Portability: CIM and UbiCollab should be deployed on ranged devices, which include PC, laptop, PDA and cell phone. Especially, run on PDA and cell phone is the pre-condition of mobility, which is required by white paper of UbiCollab.[1]

Availability: Availability relate to independency, since UbiCollab should not breakdown because of peer leave and network problems. However, this doesn¡t means all the services have to available even the specific vendor leave. I want to use a pure P2P topology in order to satisfy this requirement.

Security: Collaboration instance is the virtual space, where services can be shared from access to. We need some security protocols to prevent private services and CI groups from accessible for all peers. JXTA support authentication for that purpose.

Performance: As a groupware, UbiCollab shoud have a good performance, which gives users satisfied experience. CIM is responsible for data synchronization and CI management. CIM should replicate data to all the related peers on time with concurrency control, which make sure data consistency. The detail of how groupware should perform was mentioned in chapter two.

Scalability: CIM and UbiCollab must be scalable when new functional requirement being added in. New services could be extended in future work. We should make sure the joint is seamless. By using OSGI platform as the container, new added services with standard interface can be easily resolved and run. Pure P2P network also gives extendable feature when peers increase.

4.3 Functional requirements

I list functional requirements of CIM in table with description. Details are given at follow.

ID	Description
F1	CIM responsible for create, delete and modify CI.
F2	CIM responsible for data synchronization
F3	CIM responsible for create and manage a persistent local space for stores configure data in each UbiNode. These data used for CIM initiation and access replica data at runtime.
F4	CIM responsible for discover groups and users by assigned attribute(s).
F5	CIM communicate with applications and proxies, which access and update CIs data through CIM.
F6	CIM could register with some proxies for get context awareness information and local system change.
F7	Membership through distribution of URI.

The first two are easy understood and the data synchronization is mentioned above. For the F3, CIM maintains local folder system I mentioned in last chapter and the details will be given in next chapters. CIM use JXTA provided discovery service to lookup remote peers and CIs by specify name or other attributes (optional). As the manager of CI, all applications and proxies should access CI through CIM, CIM define the interfaces for update CI information, changed data could store in the local file system if required. F6 is similar with F5 but more specific. For example, the location change will be received by RFID reader then send to a proxy, which registered with CIM. CIM get the changed location information and use it to update local CI, in this case, the user¡s location change affect online status.

4.4 Non-functional requirement

ID	Description
N1	UbiCollab and CIM must implemented as open source
N2	CIM and UbiCollab should be implemented on rang of devices. (PDA, laptop, tablet, mobile phone)
N3	CIM and UbiCollab must satisfy mobility
N4	UbiCollab components should independent with third part components.
N5	UbiCollab, CIM and other components should be implemented in OSGI container.
N6	UbiCollab components should light weight and small size

These non-functional requirements are specify criteria that can be used to judge the operation of a system, rather than tell what CIM and UbiCollab should to do-these details behaviors are mentioned in functional requirements. The first requirement was clearly described in UbiCollab project. The code, the libraries and the platforms used to develop CIM must have open source license. N2 decide the portability quality requirement which include N3, since mobility means system should deployed on PDA and cell phone. As ubiquitous system, it should make sure the components are independent in order to treat with the dynamic characteristic of ubiquitous environment. This feature also makes system spontaneous join the ubiquitous world.[19] N5 is more details than others, but since OSGI container has different specifications, the real implementation should consider compatible. The last one caused by consider thin system,[20] which dose¡s has as enough capability and available space as laptop, such as PDA or mobile phone. For make these thin clients could use UbiCollab and CIM, the components must not as big as hundreds mega-byte and don¡t require big memory.

4.5 Hardware, software and constraints

Hardware and software which used for develop and test CIM have to be open source ones. These devices, library and platform have some constraints both due to capability and compatible. I list these required COTS components and analyze constraints after that.

Hardware:

For the requirements of mobility, PDA is required. We use HTC TyTN 2 as the testing PDA. The details of specification of it will be described in implementation chapter. We use a router to create a sub-network instead of using NTNU network since that tested has problem. For discovery service, they using a RFID reader and RFID tag (an electric rabbit). Other developing and testing tools are laptops.

Software:

The runtime and JVM, we using J9 platform which support J2ME for embedded system development, the details of it can be found in implementation chapter. The container of our modules is OSGI container, Equinox specification. I choice JXME, version 2.1.3 as the JXTA platform and library (include Log4J library). For GUI development we agree with the SWT as the toolkit.

Constraints:

The PDA has smaller screen compare with laptop and tablet. Display and control is unconvinced. The small memory is a problem also, if multi-session and modules need to be load and run, it takes long time and some times crush. The Equinox not support Swing which developed by SUN, we have to using SWT. Even it has a good performance, developing and testing has some constraints. For example, the SWT create an individual thread for display, outside thread can¡t modify display directly. It takes our some time to solve this problem. Another good OSGI container Knopflerfish is the alternative at the beginning of development phase; however, it can¡t satisfy with some purpose of session manager¡s task.

For J9 platform, it is not support enough utility libraries by itself. We have to change the library by ourselves and re-compile some source code (al of them are open source). The JXME development required rendezvous peer as the connecter of different network. I have not deployed any UbiNodes for that purpose, which could limit the communication scale of this P2P network (only in sub-network).

Chapter 5 System Architecture

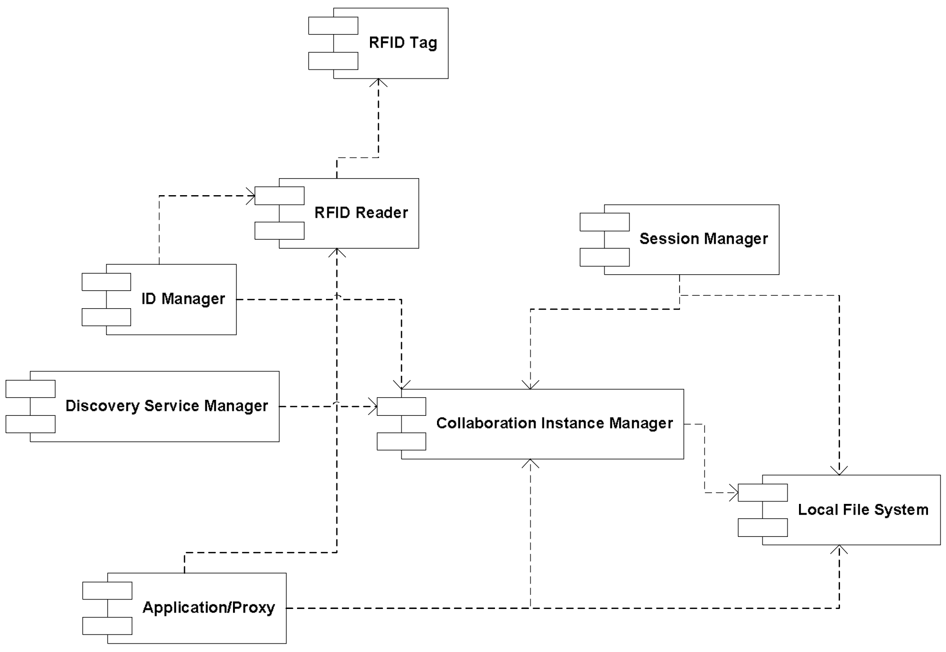
5.1 Introduction

Based on requirement specification, I design the architecture of collaboration instance and related modules. I will give the overview relationship between CIM and other module, applications. Then give the details architecture of CIM itself. These architectures are figured out from different point of view such as static view, process view and detail for specific services.

5.2 System logical static view

Firstly, I describe the relationship between CIM and other modules, since all of them contribute to UbiCollab and communicate with CIM. Figure 5.2 is the component logical view of it.

Figure 5.2 logical view of UbiCollab components.



All of other modules/components use CIM and CIM also use some of them. Session manager get update message from CIM if the local common session templet is changed. Session manager and other application/ proxy could directly modify the local files, CIM monitor local file system or get through other way to know these change, then propagate update data to other peers.

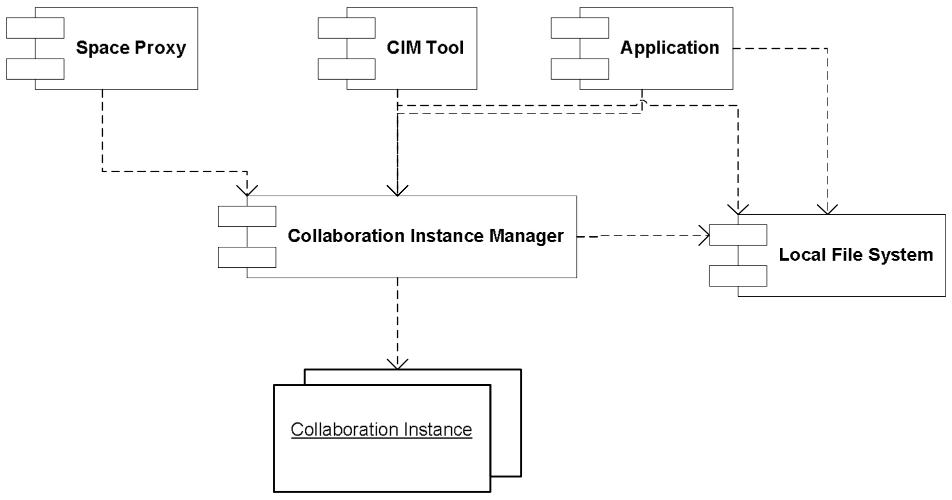
Discovery manager and ID manager publish discovered services to CIM or check imputed ID with correct one. Some application/proxy, ID manager and service discovery manager (optional) get the information from RDIF tag through RFID reader, which in our case, a RDIF pen. These information will cause CI update by calling CIM.

In real implementation, user and other modules, application/proxy use CIM directly or through a CIM tool, which support interface to them. In this diagram, I don¡t include CIM tool because most of other mentioned modules can use CIM directly.

5.3 Collaboration Instance

The main task of CIM is to manage collaboration instance. User can create and join multi CIs. However, I assume that each UbiNode can only have one active CI at one time. This assumption is due to JXTA group creation and pipe service. I use one pre-defined pipe advertisement for pipe creation and the pipe will reconfigure itself by discovery service, which specified by group. If multi groups alive at the same time, pipe don¡t know who is owner of the discovery service. This assumption is not a limitation but ease the developing, in future work, multi groups at same time is possible if corresponding pipes were created for them.

Figure 5.3: CIM, CI and related modules.



This diagram shows that all modules affect CI through CIM. Space proxy is the proxy used for sending location change message to CIM through register with CIM. CIM get information then update CI status and active CI. CIM tool and application is the interface to user, who can do change to current CI, this change update to CI through CIM.

As mentioned above, a local file system is important to my CIM research. CIM monitor file changes that could cause by application or CIM tool. Once CIM knows the change, it propagates updated data to other peers.

5.4 CIM and JXTA services

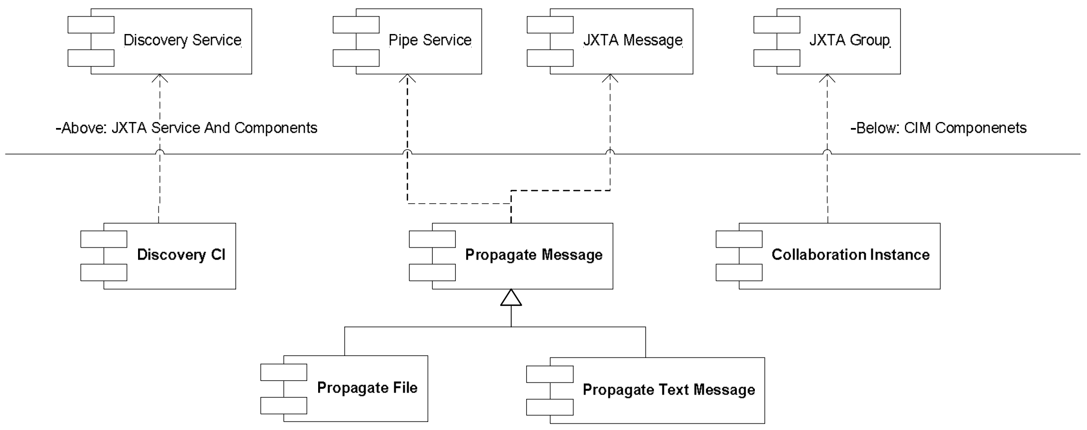
JXTA is the foundational platform used for develop P2P network and provides basic functions for CIM. Figure 5.4 is the diagram of relationship between CIM services/components and JXTA services/component.

JXTA discovery service is the service used for discovery JXTA group, publish advertisement and as the parameter of other service. CIM use it for discovery remote CI instance and other UbiNode. Pipe service supports communication channel for sending message through it. The implementation in CIM is propagate message to UbiNodes, who in the same group. Two specifications of the propagate message are: send text message, which responsible for group chatting and broadcast message; send file, which responsible for sending file content (input stream) to other UbiNodes.

Message is the basic unit for JXTA communication. Each message can have zero to more message element, who contains text string or input stream. CIM propagate message by using pipe and message.

Each JXTA group abstracted as one collaboration instance. But JXTA group only supports basic initiation of CI and a discoverable instance. Attributes and higher level implementation should be added in.

Figure 5.4: CIM services and JXTA services



5.5 Process view and MVC pattern of CIM

I figured out some static views which show relationship between modules/components. However, these diagrams didn¡t show how data transfer between them. I give the process view in order to show the concurrency and synchronization aspects of CIM and other modules.

Figure 5.5.1 shows the process of application and space proxy communicate with CIM and how CIM update CI, store update date to local file system then propagate to same group peer. CIM call space proxy, which responsible for send location change message derived from space manager or other modules. When location change event happen, space proxy callback to CIM, who firstly update information to all the CIs, then store the change to local file system, finally replicate these change to other peers.

For other applications, they directly call CIM, use it to update CI. After update, the process is same as space proxy did.

Figure 5.5.1 application and space proxy use CIM for update CI

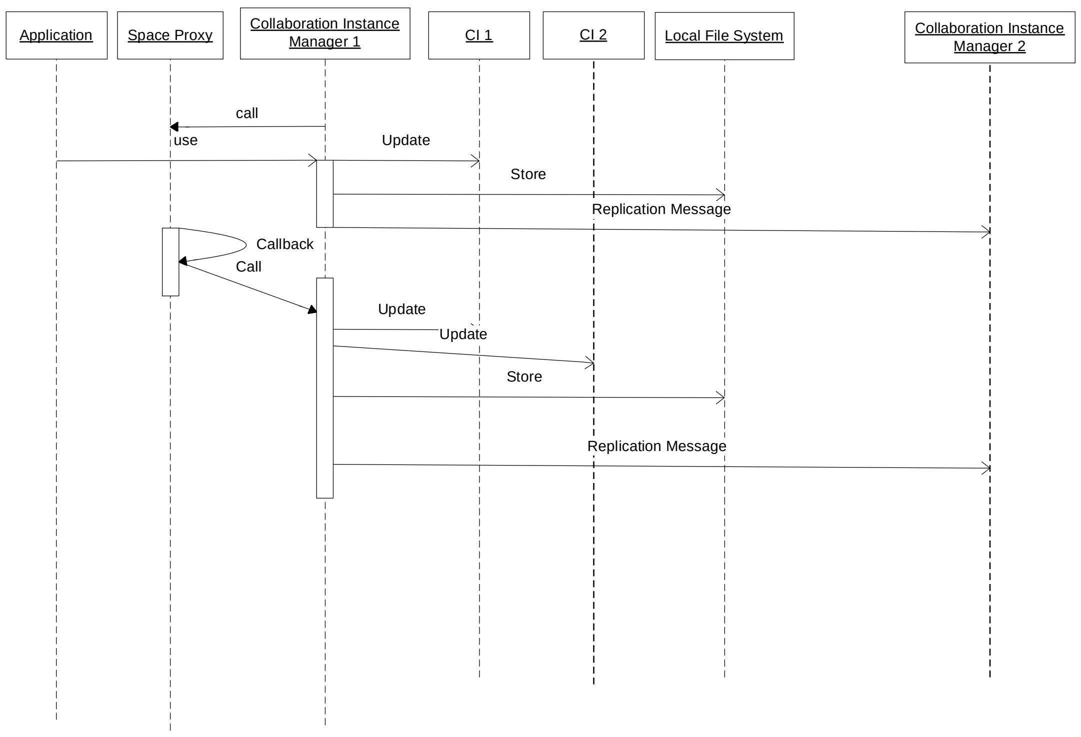
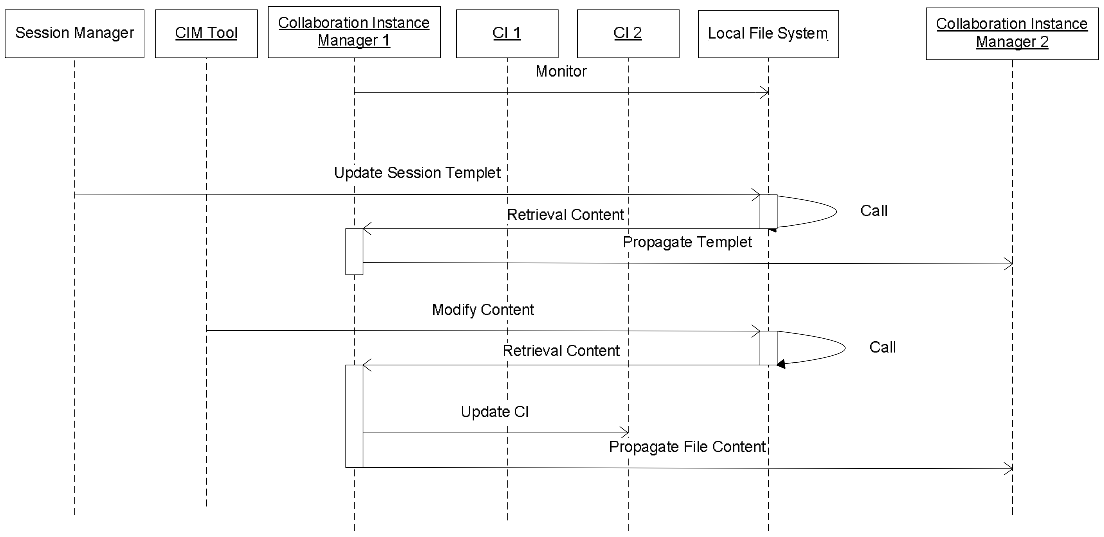


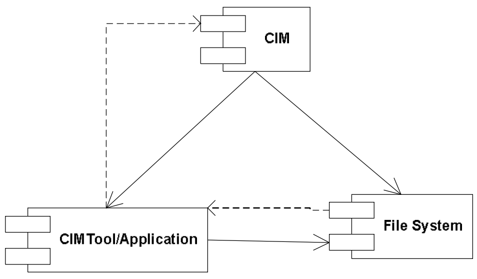
Figure 5.5.2 Session Manager and CIM too use CIM to update CI and local file system



The main approach used in my system is the file monitor process, which inspired by the file system plan B which was mentioned early. Some modules and applications directly update the file system without using CIM. Figure 5.5.2 shows how CIM treat with such kind of situation. CIM monitor file system in case files changed by applications or other modules by using file monitor system, which generates event when file content modified or new folders created. In this diagram, session manager updates common session templet and CIMTool updates file content. CIM receive event, retrieval updated content from event, and then propagate file content to other peers. CIM tool could be the GUI of CIM or other application.

Based on the process, I consider CIM system as an implementation of MVC pattern.[21] CIMTool or other application is the View, which displays the data. CIM is the Control module, responsible for treat with data in local system. Local file system doesn¡t take care of the data it stored. Figure 5.5.3 is the MVC pattern of them.

Figure 5.5.3 MVC view of CIM system



The solid lines indicate a direct association such as file write and read, and the dashed lines indicate an indirect association, which implemented as file listener and other event handler.

5.6 CIM classes

I figured out UML models from logical and process view in order to show relationship between UbiCollab system modules and CIM. In this section I give details of CIM classes and functions for implementation.

Figure 5.6: CIM and services classes.

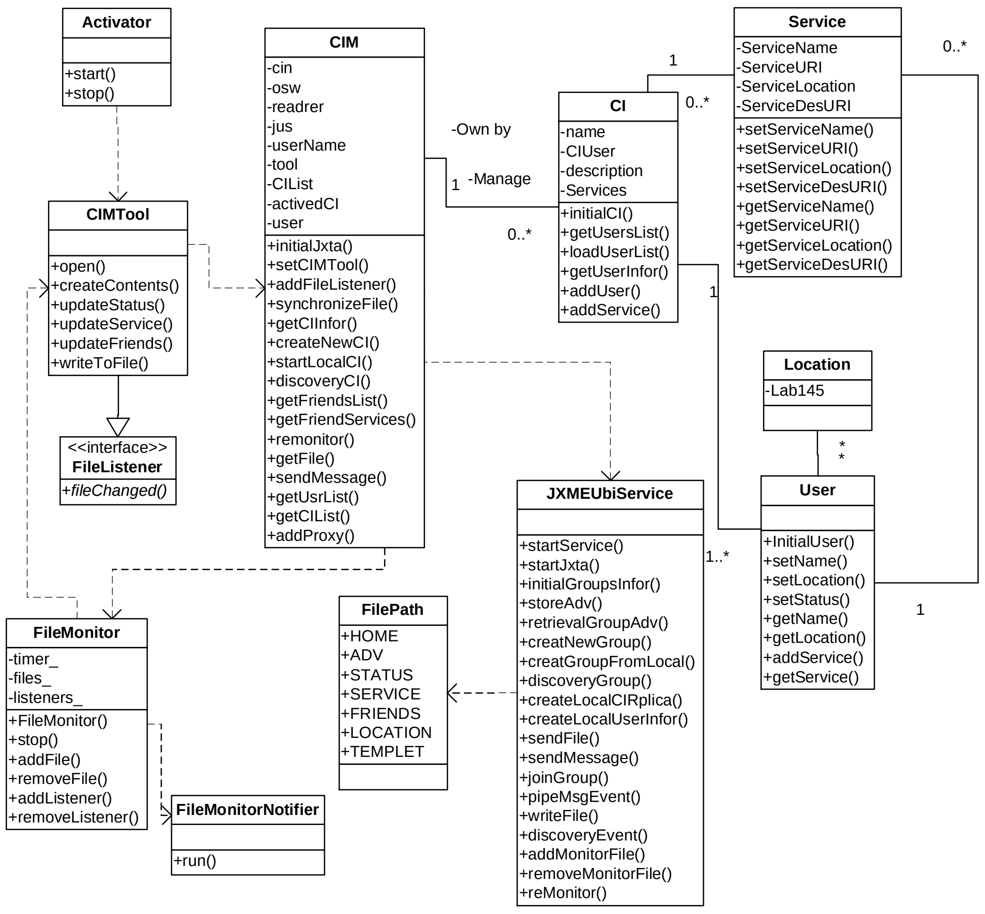


Figure 5.6 is my design of collaboration instance manager system. The two main classes for this system if JXMEUbiService and CIM. The first one responsible initial JXME network and supports basic services. The CIM is the collaboration instance manager, who uses the JXTA service and supports higher level services such as data synchronization. The CIM class manages CI, who could have Service, User. Each user has Location and Service.

File monitor service supported by class FileMonitor. It generates fileChanged event when monitored file content was changed. The listener who generates this event is the interface FileListener. Inside this interface, a method fileChanged was defined.

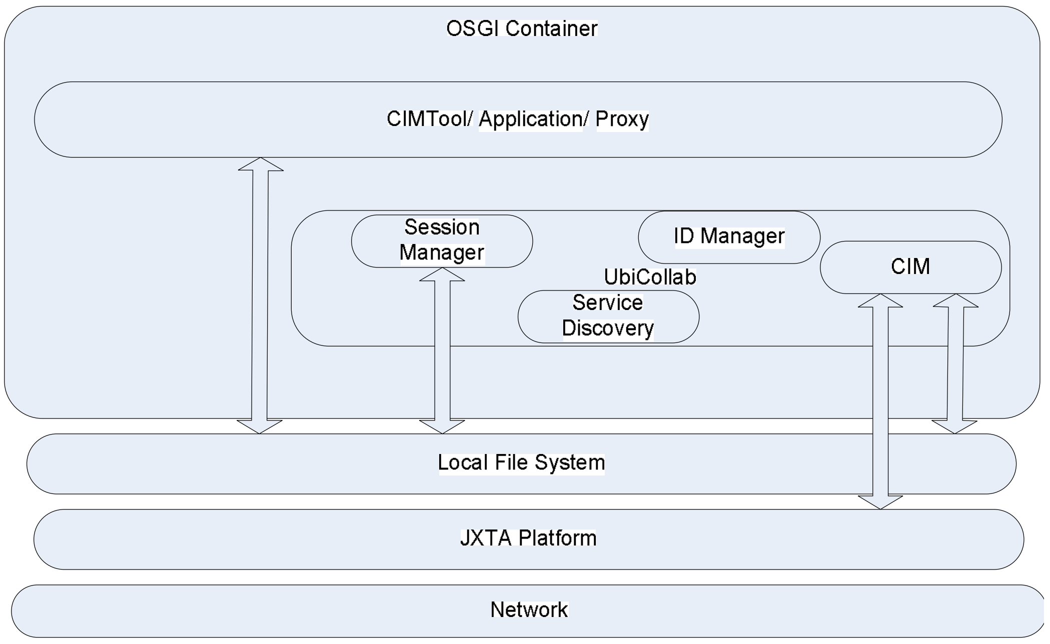
CIMTool is the graphic user interface (GUI) of this system. It also could be considered as any application who wants to use CIM. CIMTool implements the FileListener and describes what actives the system should do when file content changed. The actives such as user¡s status change, add service on GUI, add user¡s name and service when new user join current CI are based on this file monitor approach.

In a short word, the file system is the media which connected CIM and other application with each other. I will give details about file system and CIM in next chapter.

5.7 System overview

Finally, I give the system overview represented by layered services.

Figure 5.7: System overview.



CM and other UbiServices are implemented as OSGI bundles. Applications could be developed as bundles. However, outside applications should be allowed access to UbiCollab in future work.

Chapter 6 Implementation

6.1 Introduction

This chapter includes the details of implementation of CIM. Firstly I introduce the common scenario of UbiCollab project. This scenario shows usability of all modules and has specific part for CIM. Then I give the test bed for system development and testing. Before describe CIM system, I want to show the implementation change log of using JXTA and JXME libraries. This description gives the reason of why select JXME as the library and the limitation of JXME.

This chapter mainly includes the details of CIM implementation and the usability of CIM through a GUI. I want to demonstrate the data synchronization, local file system and how CIM communication with other systems.

6.2 Scenario for demonstration

The whole content of common scenario could be found in appendix A1. There are four stories, which talk about the daily life of a user of UbiCollab called ¡John¡. All of them show the utility of CIM, Session manager, ID manager and Service Discovery manager. I am not explaining the whole content of the scenario here since it is a long story and my report should only focus on CIM. However, I take a simple example from them and explicate all scenarios that relate to CIM.

John uses a RFID reader to scan the RFID tag in his office. His UbiBuddy (the device such as PDA and be installed UbiCollab platform) knows he is in office now and set him available for the CI group ¡COLLEAGUE¡. At the same time, his status should be ¡offline¡ for other groups.

John should use his to find a printer in his office by using Bluetooth or RFID reader on the device. After then, the found service should be published to CI services system and notify all of the users in the same CI. The users can access the service if they have the password. He is furthermore prompted whether he wants to change the runtime states of some of his current sessions. Specifically, the sessions that are running is a presence service and a bulletin-board service. The first shows presence related information from a CI/ group; the latter is a bulletin-board specific for a given CI/ group. Confirming this, by interacting with the UbiNode, the ongoing sessions are adapted to the current environment. After the applications are initialized, a message pops up on the bulletin board viewer, telling him that the weekly project meeting is to be held at Room 354 in one hour.

This short scenario shows some utilities of the four modules but not all. The first section is related to location awareness service supported by CIM. He can find printer service through the Service Discovery manager. Other users should access it by password authorization is supported by ID manager (password should relate to specific ID). The session manager manages the ongoing sessions.

I explain all scenario parts which related to CIM following:

1: John holds a RFID reader over a RFID tag located at the office entrance. This action positions the user to the collaboration space ¡office¡¡ and also activates the identified space. This service has been configured to modify his presence for his work-related CIs. Since he is working with a specific project he does not want to be interrupted, so he has specified that he should only be visible to the participants of the collaboration instance ¡project UbiCollab¡¡ and unavailable to the other CI groups he is a member of to avoid non-related, external annoyances.

Explication: this scenario relate to location awareness of CIM. The RFID information should be string or XML format which contains the location identifier content. A space proxy, which is a application callback to CIM when there are location update. CIM use the information updates all of the CIs. If a CI¡s pre-defined space scale contains current location identifier, this CI should be active.

2 He browses the contents of the CI/ project area through a Co-editing notepad, and notices that some of the project reports have been updated this morning.

Explication: Co-editing system relate CIM data synchronization and concurrency control. Content modification should be updated to all the UbiNodes in the same CI group. While two users update the same object, collision should happen.

3 John takes out his windows based mobile and opens the ¡Ubi Buddy¡ list. He finds ¡John¡s computer¡ from his equipment list, which is the work computer that resides in his office. At the same time, his colleague at another floor find John is online in the group ¡Colleague¡. And the services (devices) found by John¡s PDA are shown. Through his PDA, John is prompted to use the ID he had already specified for the services or create a new one. His colleague chats with him because he wants to use John¡s printer, but John tells him ¡I am going to a presentation¡.

Explication: CIM responsible for propagate local services to CI shared space. The service should be represented by URL of the address of them. If any new service been added, users have privilege to it should notified. Chatting is another important feature of UbiCollab system. CIM should provides both group chatting and one-one chatting.

4: After 10 minutes of presentation, he wants another professor to show a related document. He pushes the avatar of that professor and type ¡Hi Jane, I want you come and show the document I gave you yesterday¡ to the popup chatting dialog window. Jane is busy now, but he opens the equipment ¡projector¡ in the equipment list of John¡s and open the document. This document is show on the AUD after transfer finish.

Explication: Similar with the third one. CIM need a GUI to show the utilities. This GUI design is also important since the feeling of user should be taken into consider.

I will mention details of how CIM satisfies these scenarios in next chapters and show the GUI (CIMTool) made for use CIM.

6.3 Test bed

I list the details of devices and software platform/environment for CIM implementation and UbiCollab project.

Hardware

Tablet:

Type: ASUS R2H

Specifications:

- Intel? Celeron? M ULV Processor (900MHz)
- Genuine Windows? XP Tablet PC Edition
- Onboard 256MB, DDRII 533, 1x SoDimm socket for expansion up to 768MB DDRII 667 DRAM support
- 7" WXGA touch screen LCD, ASUS Splendid Video Intelligent Engine
- PATA 1.8" HDD 4200PRM 40 GB
- Bluetooth? V2.0 + EDR, 3x USB, 1x SD Card -Reader, 1x GPS, 1x Finger Print Reader
- 23.4 x 13.3 x 2.8cm, 830g

PDA:

Type: HTC TyTN 2

Processor: Qualcomm? MSM7200TM, 400MHz

Operating System Windows Mobile? 6 Professional

Memory: ROM: 256MB RAM: 128MB SDRAM

Network: HSDPA/UMTS: Tri-band 850, 1900, 2100 MHz HSDPA: Up to 384kbps for upload and

3.6Mbps for download
UMTS: Up to 384kbps for upload and download
GSM/GPRS/EDGE: Quad-band 850, 900, 1800, 1900 MHz (The device will operate on frequencies available from the cellular network)

Connectivity: Bluetooth? 2.0 Wi-Fi?: IEEE 802.11 b/g HTC ExtUSB? (11 -pin mini-USB and audio jack in one) GPS antenna connector

Laptop: The laptop is powerful enough for development and testing. I will not give the details of laptop here. The system used in UbiCollab and CIM is Windows XP SP2.

RFID Reader: The reader for read RFID tag. However, this part is not related to my work. I don¡t get the detail of it.

RFID tag: Unknown. .

Network:

- WIFI 802.11
- Bluetooth
- WIFI Router

Software

JRE: J9? virtual mac hine: IBM J9 optimized for supported platforms. It is already used in production and proven in independent testing to provide a fast runtime environment for embedded systems. We use j9-cdc-arm JVM for deployed on HTC PDA and use j9-cdc-x86 one for development, debug.

OSGI platform: Equinox. Equinox is an implementation of the OSGi R4 core framework specification and is the foundation of the Eclipse platform (as plugin). There is a set of bundles that implement various optional OSGi services and other infrastructure for running OSGi-based systems. The reason for using Equinox is it tested compatible with PDA and satisfy all developers¡ purpose and supported by J9. However, other OSGI specification such as Knopflerfish is still possible for the container of UbiCollab.

JXTA framework: JXME_Proxy_less (CDC) version 2.1.3. This is a stable and latest version of JXME community at this time. Tested work smoothly and compatible with J9, PDA and connected successfully to wireless network. The size of required libraries is around 1100 KB (include the Log4J, which should supported by other vendor), the pure JXME jar package is 824 KB.

6.4 From JXTA to JXME. History and change issues of the P2P network development

I have developed a prototype of CIM system before. This CIM developed by using JXTA standard version library (JXSE) and tested works well. This prototype supported basic functions of CIM such as chatting. However, these functions are low level ones, which not satisfy with high level CI management requirements. For the purpose of giving high usability, I re-develop a new CIM system.

Another reason for develop the new CIM is the mobility requirement of UbiCollab. JXSE not supports mobile device developing environment. JXME has two specifications, which are obviously different developing feeling. One of them is JXME_CDC/JXME_proxyless, which supports java ME Connected Device Configuration (CDC) framework. This framework used for build applications on embedded devices. Another specification is JXME_CLDC, which compatibles with java Connected Limited Device Configuration (CLDC).

The PDA, tablet and laptop for my development and testing are powerful enough to use the JXME_CDC library. Compare with service supporting, the CDC one are more similar with the JXSE specification than the CLDC one. In actually, it more likes the older version of JXSE. However, it has limitation such as no local cache supported and less utility classes.

JXME device commonly using wireless network and arise the edge-IP address problem. The IP addresses of them are dynamically changed. In this case, rendezvous (RDV) peers is mandatory. However, my test bed network environment is the sub-network and all the UbiNode¡s capacity is powerful enough to run the JXME_CDC_Prxyless one. As what the name stands for, it not mandatory requires RDV peer and supports most of the services of JXSE.

My work-mates and I tested the JXME_CDC connecting by using VPN in NTNU sub-network firstly, but it not work. Finally we decide to use router (AP) create a local network. Another problem is the JXME group creation approach is different with the JXME one. And some utility libraries are not compatible with it such as JXTA CMS. This library provides interface and functions for shared file management. I planed to use CMS make a shared file system. However, I have to develop a similar local file system by myself due to the incompatible.

JXME not support cache directly, which used for storing advertisements. Alternative, advertisement can be stored and retrieval by system input/output, which reads and writes XML content into a self-defined file. This advertisement file is part of my local file system.

6.5 File system for UbiCollab

Local file system is important to CIM. It not only provides a persistent local storage space but also works for CIM as the share space. CIM responsible for construct the folder structure and files inside.

Each UbiNode has the same folder structure. The root folder called CI, which you can choice where you want it to be placed. Under that are folders who represent each CI group, named by each CI group name.

Two folders inside each CI group folder, named as ¡adv¡ and ¡Myfolder¡. The adv folder store a file named ¡peerGroupAdv.xml¡, which store the JXTA group advertisement. This advertisement used for create group from local without create a new advertisement. Inside Myfoder folder is the subfolders which store CI resource. User and developer in future can self-define folders as they want. I defined 7 folders for testing.

Five folders have corresponding functions supporting in the current CIM system. They are ¡Friends¡, ¡Service¡, ¡Templet¡, ¡Status¡, and ¡Location¡. Beside ¡Friends¡ folder, others contain an xml file, which contains resource description. For example, the service.xml could contain the service name, description and URL address. It should be represent in standard XML format, however, since I have not use xml parser to read and write xml file, the actual implementation only use simple text string, each line contain predefined content. It is not affect testing quality of CIM since easy to add xml parser in future and change the format.

The Friends folder contains subfolders named by other UbiNode¡s name. These peers are in the same CI group. Each friend folder contains the same subfolder structure and namespace as Myfolder folder except the ¡Friends¡ one. User¡s resource are managed and shared through replicate the file system.

```
-CI/
------adv/PeerGroupADV.xml
------Myfolder/
        -----------Templet/templet.xml
        -----------Content/content.xml
        -----------Service/service.xml
        -----------Status/status.xml
        -----------Location/location.xml
        -----------Friends/
                   ---------Templet/templet.xml
                   -----------Content/content.xml
                   -----------Service/service.xml
                   -----------Status/status.xml
               -----------Location/location.xml
```

CIM responsible for add interested files for monitor content change. The FileMonitor and FileListener classes are open source code. I use the source code directly without make it as jar library. CIM use addMonitorFile() method of FileMonitor class to add interested file. After that, the instance of this class adds a listener responsible generate message when file content changed. In my case, the CIMTool class implements the FileListener interface and overload fileChanged(event e) method. The GUI should update display and make use of CIM to replicate data.

6.6 Synchronization of file system

Once monitored file content was changed, CIM decide how to do corresponding date synchronization. Basically, the changed file will be replicate to all the UbiNodes who are in the same CI group and place at hardcoded place. For example, user ¡Wang¡ manually set status to ¡online¡, the file ¡CI/Myfolde/Status/status.xml¡ content is overwritten to ¡online.¡ CIMTool get the update event and call CIM to distribute this file to his friends. On another side, his friend called ¡John¡ will found ¡Wang¡ status changed. CIMTool GUI updates display since it receives notification that the file ¡CI/Myfolder/Friends/Wang/Status/status.xml¡ content was changed.

CIM uses JXME propagate pipe to send file. JXTA Message is the unit for JXTA communication. Message contains MessageElement which is the real data user want to send. Message also has NameSpace, which used for distinguish different kind of message. I defined FileMessage namespace and TextMessage namespace. They separate file message from text message, which used for send chatting message.

File message contains InputStreamMessageElement, which is constructed by the file input stream. Besides the file, two StringMessageElement are added into the message object. One is the FileType and another is the SenderName. They are simple strings. The FileType will be explained later and the SerderName is the user of this UbiNode. JXME has InputPipeListener, which generates event when receive message. Because of the message is broadcasted to all peers include the sender, a SenderName string is the precondition of message parser and operation. If JXME receive the same sender name as the user, it not looks into it. This SenderName should be changed to a unique ID for security and correct reason. However, this should take ID manager into consider in the advanced version in future.

CIM receives the file message and retrievals the InputStreamMessageElement from message object. Depends on the FileType element, CIM decides where should the received file be put in. Some condition judgments take care of that.

The final process of file synchronization is use CIM to do some update work based on the changed content of file. It should be careful here is: CIM and CIMTool not directly retrieval the message element content from pipe message. They just decide where should put them into. The file monitor will tell CIM what should do on next step such as update CI classes and update GUI display.

6.7 Space proxy and location awareness

One scenario of CIM is when use enters some predefined place. He/she manually read the RFID tag by UbiNode, the space information will be transferred to a space proxy, which send message to CIM for CI update. If CIM find there is a CI¡s predefined location is equals with the received one, it changes this CI¡s status to online and set the original one ¡offline¡.

I have not defined what the proxy is, since it should communicate with ID manager and other service discovery service which not falls into my research domain. I define a interface in CI call addProxy(), it used for add any proxy including the space proxy. The difference between a proxy and a application we agreed with is: a proxy should be the applications who is run in background.

I designed the CIM add proxy by using callback approach. Transfer the reference of CIM to proxy and when proxy get some event, it callback to CIM, who does corresponding work. However, it could works in an alternative way, by using the file system. All CI groups keep a file ¡location.xml¡, which stores the predefined physical locations, where this CI can apply to. For example, CI ¡Work¡ has three locations in the file: ¡Office 145¡, ¡Lab145¡ and ¡Meeting Room 245¡. If the user enter into Lab145 and use RFID reader to read the tag, the same string in the RFID ¡Lab145¡ will be received by space proxy. Space proxy not directly talks to CIM, it writes the string into a temp file which is monitored by CIM. CIM then check the content with the location files defined by user before and update corresponding CI status.

6.8 CIMTool

CIMTool is the GUI of CIM system. It is the view of system and also plays the role of file monitor listener. I explain how to use the simple GUI to show the utility of CIM.

Initial

The main window of CIMTool is shown in figure 6.8.1. This GUI was developed by SWT: the Standard Widget Toolkit, which is an open source widget toolkit for Java designed to provide efficient, portable access to the user-interface facilities of the operating systems on which it is implemented. The reason for use that is because of J9 not support with swing and its good performance.

The top left button Start UbiCollab used for initial JXTA network and the CIM system. After that, user can create a new CI by input the CI name into the text part left to the button createNewCI. User then pushes the button for create that CI. Alternative, user can start a created CI by select the CI name from the dropdown combo and then push button ¡startLocalCI¡. Now, there are five testing CIs.

I start the local CI family for example. After start, the tree content at the left bottom is updated. There are two tested user in this group. Under each name is the sub-tree shows the services that published by this user. From figure 6.8.2, user ¡wang¡ has four services.

Figure 6.8.1 CIMTool main window

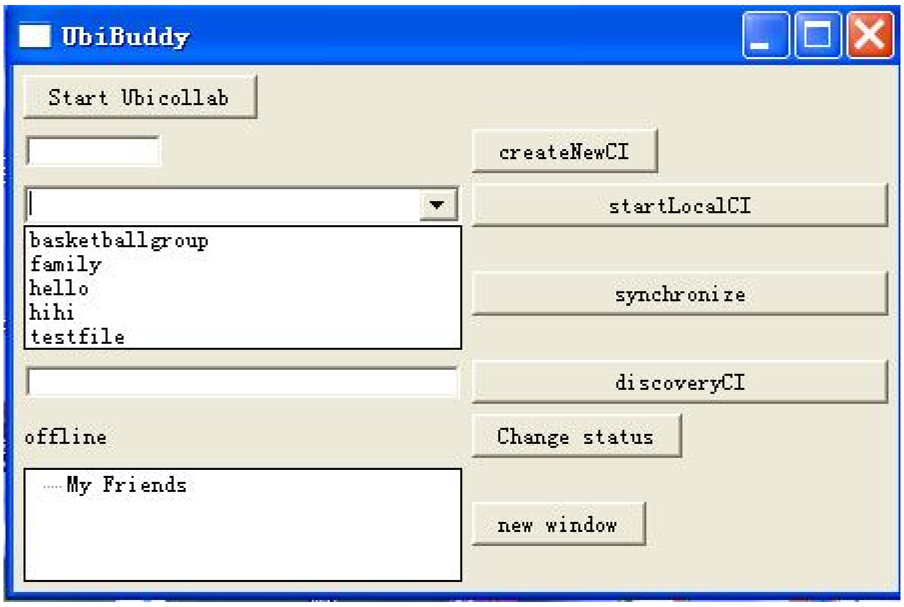
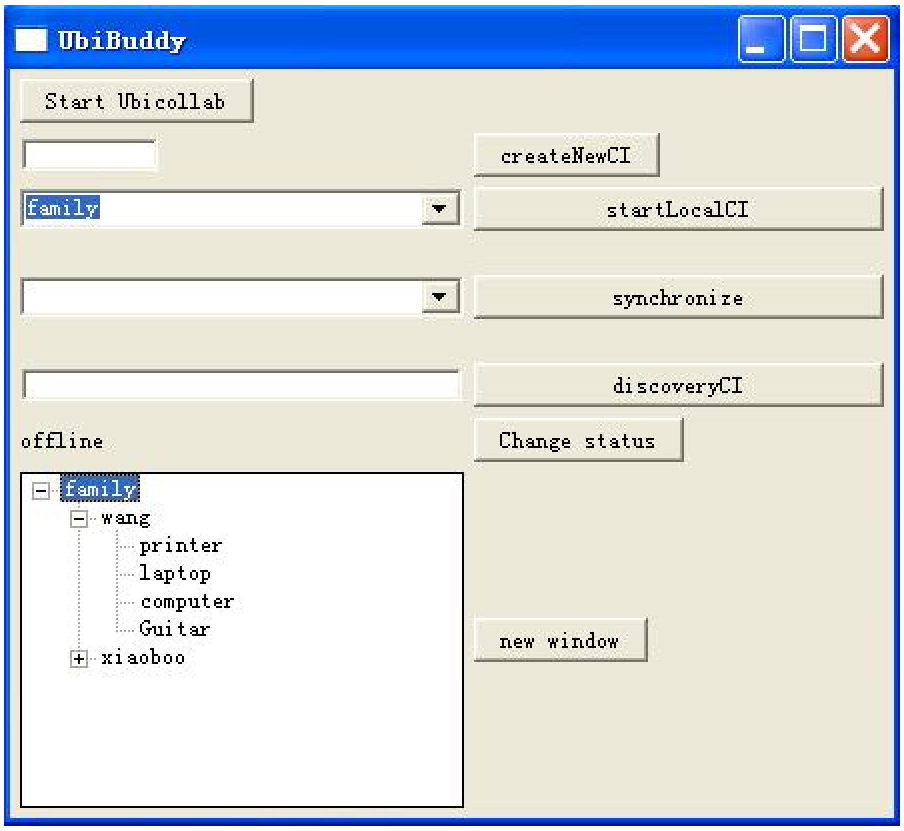


Figure 6.8.2: Start CI group ¡family¡



These user list and service list could dynamic changed due to user leaving and joining, service add and delete. These changes are updated to local display immediately based on my local file monitor approach and then synchronized to other UbiNodes based on the JXTA P2P network.

The services here are just strings that represent names of them. We have no people works on service publish and make them could be used by others. The possible way is make the names as the link to service URL address or other references that could make use of services. When click a service name, a new window popup with the information of it and makes service can be use.

The button discoveryCI used for lookup specific CI by name. After find the CI, my peer automatically joins it. The group concept in JXTA network is the instance who implements specific services and with same agreements. Join group means implement the same group services by using the same group advertisement. JXTA support a discovery service, which used for discover remote group by name or other attributes. The service also used for discovery individual peers and publish advertisement for it available for other peers. The discoverGroup(String name) method defined in CIM is the implementation of discovery service.

Status change

The button changeStatus is used for manually changes my online status and test the status synchronization. When push this button, my status changed from offline to online and verse vice. These changes are base on local file status.xml. Current status are overwritten to this file, CIM monitor this file and change current user¡s status then propagate to other UbiNodes. This status change could also invoked by location awareness, if the space proxy sends the space change message, current CI could disappear from current view and a new CI instead of it. At the same time other peers¡ friends¡ statuses will be changed also.

Synchronize file

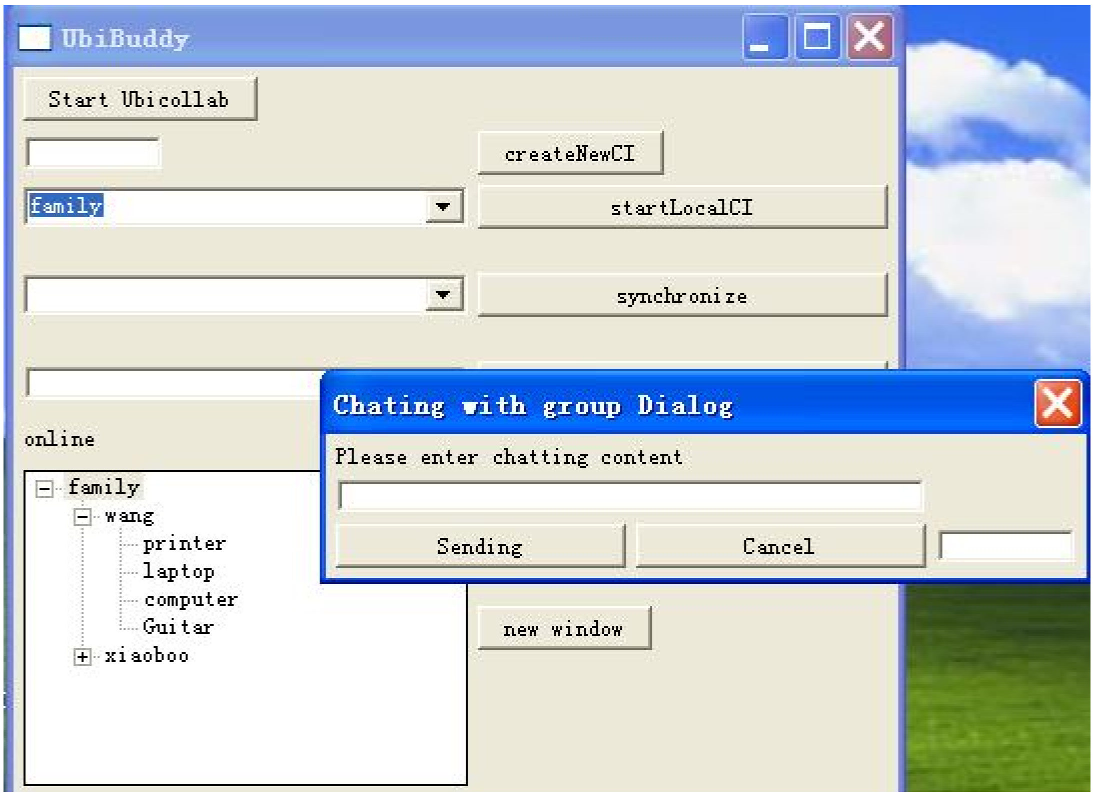
Button synchronize is used for testing file synchronize manually. From the dropdown combo, user could choice the file that he/she wants to propagate to others. This manually way can be easily adapted to an interface to other applications who want to make use of CIM. The method in CIM for this purpose is sendFile(String type).

Group chatting

The new window button used for popup a new dialog window for chatting. Figure 6.8.3 is the popup one. This simple one is tested for group chatting. The chatting is broadcast message which broadcast to all the peers in the same CI. User can type message and click the send button. It displays the received message with the format: username> message on another test filed.

I haven¡t implement an unicast one to one chatting method. The reasons are, firstly, I have developed the unicast chatting method in my last JXSE prototype CIM. Secondly, for develop that unicast JXTA pipe, JXTA service need to create dynamic pipe or predefine some pipes for that purpose. This unicast pipes testing take too much time for testing, this JXTA network issue could be improved in future but not my current main task.

Figure 6.8.3: group chatting dialog window:



It is hard to show the dynamic update and synchronization from picture. As mentioned above, all dynamic changed update immediately in local without waiting confirm from remote sites. It is consider the user¡s felling, no people want to wait 10 seconds from sending message or add a new service. These changes write to local file as resource. And propagate to others later. The details of code you can find in appendix A2.

Chapter 7 Evaluation

I evaluate my CIM system based on two criteria. First is how it satisfies with requirements specification, which is focus on system quality. Another is focus on how it satisfies the scenarios, which focus on utility and user¡s feeling.

Requirements specification analysis

Start from non-functional requirements. N1 is satisfied since all the develop tool and platform we used in this project are open source software such as J9, JXME and some utility libraries. N2 is achieved by using JXME and J9, which support J2ME developing. The CIM and other modules are tested work smoothly on PDA and other advanced devices. However, we can¡t deploy it on cell phone at this moment both due to JXTA limitation and compatible.

For N3, the mobility is satisfied by deploying CIM and UbiCollab on PDA and using wireless network. N4 is achieved since all the components we developed are open source, platform independent and is P2P system. Since OSGI container is the test bed for UbiCollab, N5 is no satisfied. Finally, the bundle (only for CIM system) is less than 2 MB, PDA has enough space to install it.

The quality driver (quality requirements) analysis

Portability: UbiCollab and my CIM system is platform independent and tested work on PDA and tablet, PC. (not deployed on cell phone)

Availability: It constructed by a pure P2P network, less independent means high availability.

Security: For whole UbiCollab system, security should achieved by ID manager, which not my work. For CIM, security should achieved by JXTA security protocol, which not available in current version due to time limitation and technical problem.

Performance: Data synchronization is very fast, no long time latency. But have not tested for big data stream since JXME don¡t compatible with JXTA CMS and the file transfer limitation is 8 KB for each time.

Scalability: OSGI container used for this purpose. New services add without reboot system.

Functional requirements analysis

CIM support methods for F1 and F2. F3 is the most problem in early develop phase, since JXME not directly support cache and not find good way to solve CI information persistent storage. These problems are overcame by using file system and file monitor callback.

F4 is satisfied by discovery service. For F5, I define the interface for add proxy and application, but not assign concrete task to it since we don¡t have test object (application, proxy). But the interface is enough for future extension. The other two ones are partial satisfied. F6 is similar with F5 and F7 partially falls into ID manager¡s work. I provide the data synchronization channel for satisfy this purpose.

Scenario criteria analysis

Scenario 1: The first part of this scenario related to service discovery manager. I don¡t know the details of RFID reader. However, I define the interface in CIM for location awareness. If the space proxy gives location information, CIM should set current CI as predefined ones depends on location.

1: assume that at the same time, only one CI is active. This assumption ensures CI avoids annoyances from other.

2: I have not developed a specific Co-editing notepad application. However, a simple shared notepad is achieved through using local file system, you can edit specific file by windows editor. When you save the file, change will be synchronized to others.

3: When user¡s service discovery service finds services available around physical environment, it publishes service name and address to file system, in service.xml file. CIM then propagate the services to friends in the same group. User can chatting with other participant through group chatting.

4: We don¡t have testing service but the propagated service should include the available address of project and printer. The detail of service invoke not falls into my research area.

I give a conclusion of CIM system quality. The advantages of my CIM system are:

1: pure P2P groupware system.

2: deployed on mobile device.

3: platform independent, has big extension space.

4: File system is not a creative by me, but I make file system work for data synchronization and groupware. The simply structure and no special requirement of file system make it can be improved and adapt in different ways. For less resource and web service approach, file content is easy displayed on webpage and edited by any text editor.

The shortcomings are:

1: No authorization available at this moment, CI group is visible to all peers.

2: Only work in sub-network.

3: Not all CIM functions are supported. I planed to develop a Co-edit application but not finished due to time limitation. However, as I mentioned above, the file system provide possible to do that.

4: GUI looks not nice. We have another version UbiBuddy, which looks better than mine. This GUI just used for my purpose of testing CIMTool.

Chapter 8 Future work

Based on what I have done and take consider of the shortcomings and advantages of CIM system, I list what should be improved in future work.

Functions: more high level functions should be added in future. Such as online meeting, (audio, vide). JXME might be a boundary of that since the limitation of JXME compatible and device capacity. An alternative way is develop different specifications for ranged devices. For PC, JXSE can be used as the platform, which provides more utilities.

Shared file: even file system support a shared file environment, but the limitation of 8KB of file transfer makes its hard to share really useful content such as PDF or WORD document. CMS is a way to solve that problem if the compatible of JXME can be satisfied in future.

Rendezvous peer: rendezvous peer should be introduced into UbiCollab. These specific peers facilitate resource discovery and across the boundary of edge peer and firewall, NAT. By using that, the runtime network be extended to whole internet but not the sub-network. I don¡t think introduce RDV peer take the risk of break pure P2P architecture if we consider it as partial P2P, which belong to P2P architecture.[16] In actually, pure P2P architecture is hard to implement for a real usable groupware. The role of RDV peer could be played by any UbiNode if this peer has enough capacity. In this way, RDV peer can be changed and if there enough number of peers in UbiCollab network, it has high availability quality and no server dependency problem.

User interface design: GUI of CIM system and the UbiBuddy should be improved for give user good feeling and performance.

Chapter 9 Conclusion

I give conclusion of my research and work of CIM system based on system quality and evaluation. This CIM system achieves most purposes from stakeholder¡s point of view and is the start point of future work. Because of time limitation and technical problems such as compatible, CIM¡s functions and utility are limited. However, it should be consider as a prototype of UbiCollab. I define some interface for future applications to use CIM. It is easy to extend.

The most important features of my CIM system are, firstly, it is a pure P2P groupware based on JXME platform satisfy the purpose of mobility. Secondly, the file system gives big extension space and easy to adapt for different purpose. For web service, xml files are easy to display and parser the content. For co-working application, xml file are easy to edit with any text editor. CIM manage synchronization issues of this file system, lose coupling with applications. This feature makes development and communication with other modules very easy. Thirdly, UbiCollab and CIM system is open source and platform independent, which make it without the problem of re-development for different runtime and platform. User can self-develop interesting service or application as bundle. OSGI platform make service publish and interactive easy. This feature is also important for UbiCollab extension.

UbiCollab is a good research topic which focuses on ubiquitous and groupware system. We want people can seamlessly interact with each other and participant in natural way. CIM manages one of the most important services of UbiCollab system, data synchronization. Through JXTA P2P network and local file system management, resource synchronization is worked in an easy way and has potential of adapt to different application and platform.

There are still some shortcomings and unresolved problems need to be improved in future. I have not given the concurrency control police since time limitation and technical problems. This service is important to co-editing application. The four modules of UbiCollab have not connected together yet. In CIM, I have defined some interfaces for them.

Appendix

A 1 Common scenario

Changelog:

Date	Author	Comments
12.02.2008	Simon, Xiaobo, Xiaozhou, Kai Arne	Had a meeting about our shared scenarios. Xiaozhou had prepared a scenario.
12.02.2008	Xiaobo	Initial version. Added scenario and requirements.
13.02.2008	Xiaozhou	Added scenario
14.02.2008	Kaiarneg	Added scenes and refactored the document. Also added some comments. Added appendix (draft).
14.02.2008	Xiaozhou	Added 3 pictures
15.02.2008	Kai Arne	Merged scenes 1 and 2. Cleaned up some scenes. Deleted some of the redundant parts

This scenario is used for the UbiCollab project demo.

Scenario
Scene 1

Some services:
- Calendar service
- Mail application
- Presence viewer
- MessageViewer/ bulletinBoard/ sharedTable
    - Brainstorming application
    - Bruk av touchscreen for meldingene.

Text

John is a professor, and he is a UbiBuddy user. One day, John entered his office as usual, scanned the tag by using his Ubinode. A rabbit in his office is awoken. Right after he scan the tag and his Ubibuddy status changed to online under the group ¡office¡. A group of services showed up like light, printer and so on. He renamed his printer to ¡Printer_john_office¡ and published it on the ¡Office¡ group. He turns on his X10-enabled lamp. This service has been configured to modify his presence for his work-related CIs. Since he is working with a specific project he does not want to be interrupted, so he has specified that he should only be visible to the participants of the collaboration instance ``project x¡¡ and unavailable to the other groups/ CIs he is a member of to avoid non-related, external annoyances.

In addition, he holds a RFID reader over a RFID tag located at the office entrance. This action positions the user to the collaboration space ``office¡¡ and also activates the identified space. He is furthermore prompted whether he wants to change the runtime states of some of his current sessions. Specifically, the sessions that are running is a presence service and a bulletin-board service. The first shows presence related information from a CI/ group; the latter is a bulletin-board specific for a given CI/ group. Confirming this, by interacting with the UbiNode, the ongoing sessions are adapted to the current environment. After the applications are initialized, a message pops up on the bulletin board viewer, telling him that the weekly project meeting is to be held at Room 354 in one hour.

He browses the contents of the CI/ project area, and notices that some of the project reports have been updated this morning. However, John is left puzzled by some of the conclusions as they seem to be in contrast with some of the pending work he is currently looking into. The project is near a milestone, so the issues should be clarified as soon as possible.

He prints the report and reads it more carefully. The printer is chosen automatically based on his current location. An interface towards this service is provided through a printer control widget, allowing operations such as changing preferences, number of copies, etc. This may also be used as a service from other applications.

In order to have something to discuss at the meeting, he prepares a presentation of his current work. 10 minutes before the meeting he notices at the periphery of his attention that participants of the CI/ group starts disappearing. He finishes his presentation, and heads off to the meeting room, bringing his UbiNode.

Upon leaving the room, the session is reconfigured such that the output resources, i.e. the display on the desk and the calendar device, are disconnected and only the calendar and mail application is running.

Analysis

Storyboard text	What it means in terms of architecture	What subsystems are involved	How is the flow	What are the involved applications and services
He is currently logged into UbiBuddy.	The user logins using their real user names only known by the ID manager. In the different collaboration instances, the user should be identified by their chosen	ID Manager User Manager	Enter username and password Authentication Authorization	– Login GUI –

Storyboard text	What it means in terms of architecture	What subsystems are involved	How is the flow	What are the involved applications and services
	pseudonyms.			
¡¡he has specified that he should only be visible to the participants of the collaboration instance ``project x¡¡ and unavailable to the other CIs he is a member of.¡	- That a CI can have information about inhabitant online status. - That this information can be different from one CI to another. - That the user can decide to set this information for each CI individually. - That other inhabitants can see this information. Hva med ruting av informasjon mellom noder? Er dette	- CI manager. - Session Management if the online information will be displayed in other nodes in the Space.	Related use cases: - Create CI. - Browse CI. - Display CI. - Set CI information. - Set up session. -	- CI browser. - CI viewer. - CI information editor. - Presence display service. -

Storyboard text	What it means in terms of architecture	What subsystems are involved	How is the flow	What are the involved applications and services
	sesjonens job? If the online information is to be displayed at other nodes etc.			
Bootstrapping	The platform and application components are configured and started up	service domain session manager		
As he enters his office, he holds a RFID reader over a RFID tag located at the office entrance. This action positions the user to the collaboration space ``office¡¡ and also activates the identified space.	That services and configurations are location-based/ related to a specific CS Certain components are location based/ enabled/ supported Should the location perhaps be published? (if the user wants to?) This is an important issue for mobile users; where are mobile users located and so	CS manager Collaboration instance	Specify location Look up location	CS UI for manual location specification (if RFID is not available) CS Management Tool (tool using the rfid reader and updating location in a given CI)
¡He is [¡] prompted whether he wants to change the runtime states of some of his current sessions. [¡] the ongoing	That applications are run on the platform, and that these may be controlled by the user (run, terminate, change state etc) through the application That applications may	Service Domain Session Manager	Browse sessions Control session	Presence service Bulletin-board service Session control UI / Configure session (choose templates)

Storyboard text	What it means in terms of architecture	What subsystems are involved	How is the flow	What are the involved applications and services
sessions are adapted to the current environment. This space and how the environment, along with the services, etc is to behave, has been previously defined and configured.¡	have different configurations and that these configurations are used to set up the environments/ platform keeps configurations for different locations Services may have preferences, and these are specified through the session templates (perhaps ID manager? Does it make sense to relate different preferences based on the id used?) That sessions have different runtime states That the state of services may be kept by the platform, enabling continuity in the work for the user (if the services support this). That the configuration process may be context aware (for example, if I¡m in room A, then x, y, z should be activated automatically¡			

Storyboard text	What it means in terms of architecture	What subsystems are involved	How is the flow	What are the involved applications and services
	If I leave the room, the session should be put to a background state). That the user also should be in control of the runtime environment (browsing, initiating, terminating) Sessions may be reconfigured These configuration parameters are specified through two types of templates; session and composition. Templates, at least at the level of compositions, should be able to be shared. Session templates contain certain preferences, and may not be sharable (as they might have privacy issues the users don¡t think of etc? Are to be treated as individual items) The service proxies must implement suitable interfaces (such as for session management,			

Storyboard text	What it means in terms of architecture	What subsystems are involved	How is the flow	What are the involved applications and services
	start, stop etc).			
He browses the contents of the CI/ project area, and notices that some of the project reports have been updated this morning.	CI works as a shared file area LOOK AT COMMENT: files are represented by ¡ files or urls?	Collaboration Instance		
He prints the report, and reads it more carefully	That services are accessible from and handled by the platform That the application is able to utilize these services, both from an application and as a standalone application (e.g. print from an application, and a standalone print-control) That users should be able to control services in some way (e.g. service ui) These are furthermore appropriately configured (minimum effort needed)	Session manager Service domain (lookup)	Browse applications (needs to specify that he wants to use this app) Set up session Terminate session	Printer control UI/ control widget Printer Service Portal?
¡the session is reconfigured such that the output resources, i.e. the display on the desk and the calendar	Sessions are reconfigurable (re)configuration happens in a way that makes it possible to reinstate the session with other (compatible) services. May	Session manager Service domain Space Manager (to denote that the user is on the move) Collaboration instance (change availability of	Control session (change state) Reconfigure session Manage resources (resource management)	Session control UI

Storyboard text	What it means in terms of architecture	What subsystems are involved	How is the flow	What are the involved applications and services
device, are disconnected and only the calendar and mail application is running.	not be true for all sessions; services must themselves denote what operations are supported. Any shared services are made unavailable when a user exits the room these services are related to. EXCEPTIONS could be specified? The platform must be able to notice that the user has left the room	shared services)	Look up services	

Scene 2

Text

¡John take out his windows based mobile and opens the ¡Ubi Buddy¡ list. He finds ¡John¡s computer¡ from his equipment list, which is the work computer resides in his office. He dose¡s need a password to do this since his pda and the ¡John¡s computer¡ are in the same category ¡John¡s equipments¡. There are many other services (devices) in this category like ¡office light¡, ¡office print¡. (Figure 1)

At the same time, his colleague at another floor find John is online in the group ¡Colleague¡. (Figure 2)And the services (devices) found by John¡s PDA are shown. His colleague chats with him because he want to use John¡s printer, but John talk to him ¡I am going to a presentation¡.

[vurder ? kutte ut denne] It takes 3 minutes for John walk to the door of AUD 5. He want take a cup of coffee. He open his PDA and find the coffee machine service is found. He choose a cupchino. (The coffee machine out the door of AUD 5 is belong to a public group, so every one can access to).

Analysis

Storyboard text	What it means in terms of	What subsystems are involved	How is the flow	What are the involved

	architecture			applications and services
¡At the same time, his colleague at another floor find John is online in the group ¡Colleague¡. (Figure 2)And the services (devices) found by John¡s PDA are shown. His colleague chats with him because he want to use John¡s printer, but John talk to him ¡I am going to a presentation¡. ¡	That users are able to view contextual information about participants (services, etc). Shared services may belong to different classes: - Services to be available all the time (e.g. filesharing) - Should be available in the context of the user (e.g. location or time). - That the user carries with them (what Babak referred to as wearables) \cite{someCiting}	Collaboration instance Manager Session manager	Context awareness (location based) Look up user Look up services Resource management	Context group Context shared devices list

Scene 3
Text
[kan whiteboard ogs? brukes til noe ifm sesjonsh?ndtering (forst?tt som sporing av brukere¡s handlinger?]

After John enter the AUD 5, the devices in the AUD are found and show on the list. (Other services shown before such as coffee machine and Office computer (and are not private) are disable and disappear from the equipment list). (Figure 3)

Distribuert sesjon som involverer andre (ubi)noder, ikke bare tjenestenoder. Remote participants deltar via videokonferanse. Siden prosjektet har folk som er lokalisert p? andre sites, vil disse delta p? m?tet over videokonferanse. John invokerer en presentasjonsapplikasjon for dette scenariet.

Vil han se delte ressurser? Is?fall kan han benytte 2 viewers... sin egen og den p? remote site. Personene som joiner vil ha utstyr tilgjengelig for seg/ i sitt rom... John kan starte opp, invitere folka p? remote site, sl? opp p? deres tjenester, og velge ? benytte noe av deres utstyr/ synkronisere sesjonene. Vil strengt talt kun v?re ¨n sesjon her... Johns (han vil iallefall v?re master). De andre vil innvolveres med sitt utstyr¡? De kunne ogs? ha startet egne, kompatible sesjoner, og synkronisert disse.

At the conference room, John uses a presentation service for showing his presentation. This service also requires the use of a controller. The UbiNode itself offers a simple control service able to control the presentation. [I need to add a section here (Kai); the PDA that is running the platform also has a controller-service installed, so he uses it for controlling the presentation)].

After 10 minutes of presentation, he want another professor show an related document. He click the avatar of that professor and type ¡hi Jane, I want you come and show the document I gave you yesterday¡. Jane are busy now, but he open the equipment ¡projector¡ in the equipment list of John¡s and open the document. This document are show on the AUD after transfer finish.

Analysis

Storyboard text	What it means in terms of architecture	What subsystems are involved	How is the flow	What are the involved applications and services
At the conference room, John uses a presentation service for showing his presentation. This service also requires the use of a controller. The UbiNode itself offers a simple control service able to control the presentation.	Hvordan tilbys disse applikasjonene? Er det noe han har installert? Noe han kan sl? opp? Noe han kan dele? Instansene som inng?r her¡ composition template	Session management Service domain/ collaboration instance (implicitly, service lookup)	Browse applications, and activate template Session set up	Presentation service Presentation Controller Session control (invite, look at services the other site has available etc)
	With remote participants:			

	Sessions may involve distributed nodes/ physical distribution (in a collaborative sense). Actions reflected at one service must thus be propagated to the other participants (more trad. Cscw session management). The human grid may have different constellations; the session concept must be able to accommodate all of these.			

Scene 4

Text

Towards the end of the meeting, a discussion about the exclusion of some product features arises between Alice and John. The discussion continues until they are disrupted by another group having reserved the conference room. John proposes to continue their discussion from his office.

John and Alice heads out to the hallway. While specifying that the session should be moved, he notices that Bob, one of the key members of the product group under discussion, is at his office. To settle the discussion, he wishes to supplement with his viewpoints and asks if Bob accepts. Having shared some of his resources to UbiCollab, John specifies to his UbiNode that the current meeting session should not stop but be moved to Bob's office and adopted to the resources available.

After the discussion, John heads back to the office. He then reinstates his mail- and calendar session.

Analysis

Storyboard text	What it means in terms of architecture	What subsystems are involved	How is the flow	What are the involved applications and services

TBA: Collaboration instance/ replication

When one get the service, the system invoking the Ubi Instance manager and add the service under the user. (only an announcement for testing). The instance manager simple propagates the information to the others who subscribe to the change (the UbiNode in the same group).

The propagation should be finished in an acceptable time. (2 seconds for example).

A 2 Code

Package service:

Class jxmeUbiService

```
package service;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.List;

import fileMonitor.*;

import tool.CIMTool;

import net.jxta.discovery.DiscoveryEvent;
import net.jxta.discovery.DiscoveryListener;
import net.jxta.discovery.DiscoveryService;
import net.jxta.document.Advertisement;
import net.jxta.document.AdvertisementFactory;
import net.jxta.document.Document;
import net.jxta.document.MimeMediaType;
import net.jxta.endpoint.InputStreamMessageElement;
import net.jxta.endpoint.Message;
import net.jxta.endpoint.MessageElement;
import net.jxta.endpoint.StringMessageElement;
import net.jxta.endpoint.Message.ElementIterator;
import net.jxta.exception.PeerGroupException;
import net.jxta.id.IDFactory;
import net.jxta.peergroup.PeerGroup;
import net.jxta.peergroup.PeerGroupFactory;
import net.jxta.peergroup.PeerGroupID;
import net.jxta.pipe.InputPipe;
import net.jxta.pipe.OutputPipe;
import net.jxta.pipe.PipeID;
import net.jxta.pipe.PipeMsgEvent;
import net.jxta.pipe.PipeMsgListener;
import net.jxta.pipe.PipeService;
import net.jxta.platform.ConfigurationFactory;
import net.jxta.protocol.DiscoveryResponseMsg;
import net.jxta.protocol.ModuleImplAdvertisement;
import net.jxta.protocol.PeerGroupAdvertisement;
import net.jxta.protocol.PipeAdvertisement;
import net.jxta.rendezvous.RendezVousService;

public class jxmeUbiService implements DiscoveryListener, PipeMsgListener {

    static PeerGroup netPeerGroup = null;

    private final static String SenderMessage = "JxtaTalkSenderMessage";
    private static final String SenderName = "JxtaTalkSenderName";
    private static final String SENDERGROUPNAME = "GrpName";
    private static final String messageNameSpace = "Message";
    private static final String fileNameSpace = "Filemessage";
    private static final String fileType = "fileType";
    private static final String LocationFile = "Location";
    private static final String StatusFile = "Status";
    private static final String ServiceFile = "Service";
    private static final String ContentFile = "Content";
    private static final String TempletFile = "Templet";

    private String[] group = null;
    private String[] friendsList = null;
    private String currentGroupName = null;
    private PeerGroup mygroup = null;
    private PipeService pipeService;
    private RendezVousService rendezvous;
    private PipeAdvertisement pipeAdv;
    private InputPipe input = null;
    private OutputPipe output = null;
    private DiscoveryService discovery = null;
    private OutputStream os = null;
    private boolean joind = false;
    private File currentGroup = new File(FilePath.HOME + currentGroupName);
    private File Name = new File(currentGroup, FilePath.NAME);
    private File Location = new File(currentGroup, FilePath.LOCATION);
    private File Status = new File(currentGroup, FilePath.STATUS);
    private File Content = new File(currentGroup, FilePath.CONTENT);
    private File Friends = new File(currentGroup, FilePath.FRIENDS);
    private File Service = new File(currentGroup, FilePath.SERVICE);
    private File Templet = new File(currentGroup, FilePath.TEMPLET);
    private File NameF = new File(Name, "name.xml");
    public File LocationF = new File(Location, "location.xml");
    public File StatusF = new File(Status, "status.xml");
    public File ContentF = new File(Content, "content.xml");
    public File ServiceF = new File(Service, "service.xml");
    public File TempletF = new File(Templet, "templet.xml");
    private File advPath = new File(FilePath.HOME + currentGroupName
                + FilePath.ADV + "peerGroupAdv.xml");
    private File advF = new File(advPath, "peerGroupAdv.xml");
    private String peerName = null;
    static boolean find = false;
    private boolean monitorFile = false;
    public FileMonitor monitor;
    public CIMTool tool;

    /*
     * this function uses for initial the basic service of jxta. Discovery
     * service used to discovery other peers and group
     *
     * pipe service used to propagate or unicast message to other peer(s).
     */
    public void startService() throws IOException, InterruptedException {

        discovery = netPeerGroup.getDiscoveryService();
        // discovery.remotePublish(netPeerGroup.getPeerAdvertisement());
        discovery.addDiscoveryListener(this);
        pipeService = netPeerGroup.getPipeService();
        // Create the input pipe with this app as the message listener for this
```

```
        // pipe
        input = pipeService.createInputPipe(getMyJxtaPipeAdv(), this);
        // This pipe is a propagated pipe, therefore also bind to it
        output = pipeService.createOutputPipe(getMyJxtaPipeAdv(), 100);
        // Announce our presence
        sendMessage("Hello Ad-Hoc World ");

        // discoveryGroup("xiaobo test group");
        monitor = new FileMonitor(1000);
}

public void addListener(FileListener listener) {
        monitor.addListener(tool);
}

/**
  * Starts jxta
  *
  * @throws InterruptedException
  *
  */
public void startJxta(String name, int Portnumber) throws IOException,
                InterruptedException {
        try {
                // Set the peer name
                // System.out.println("set your peer name");
                ConfigurationFactory.setName(name);
                peerName = name;
                ConfigurationFactory.setTCPPortRange(Portnumber, Portnumber + 10);
                // ConfigurationFactory.setTcpPort(9920);
                // Configure the platform
                Advertisement config = ConfigurationFactory.newPlatformConfig();
                // save it in the default directory $cwd/.jxta
                ConfigurationFactory.save(config, false);

                // create, and Start the default jxta NetPeerGroup
                netPeerGroup = PeerGroupFactory.newNetPeerGroup();

        } catch (PeerGroupException e) {
                // could not instantiate the group, print the stack and exit
                System.out.println("fatal error : group creation failure");
                e.printStackTrace();
                System.exit(1);
        }
```

```
}

private void freshFilePath() {
        currentGroup = new File(FilePath.HOME + currentGroupName);
        Name = new File(currentGroup, FilePath.NAME);
        Location = new File(currentGroup, FilePath.LOCATION);
        Status = new File(currentGroup, FilePath.STATUS);
        Content = new File(currentGroup, FilePath.CONTENT);
        Friends = new File(currentGroup, FilePath.FRIENDS);
        Service = new File(currentGroup, FilePath.SERVICE);
        Templet = new File(currentGroup, FilePath.TEMPLET);
        NameF = new File(Name, "name.xml");
        LocationF = new File(Location, "location.xml");
        StatusF = new File(Status, "status.xml");
        ContentF = new File(Content, "content.xml");
        ServiceF = new File(Service, "service.xml");
        TempletF = new File(Templet, "templet.xml");
        advPath = new File(FilePath.HOME + currentGroupName + FilePath.ADV);
        advF = new File(advPath, "peerGroupAdv.xml");

}

public String[] getFriendServices(String name) {
        File friServices = new File(Friends, "/" + name
                        + "/Service/service.xml");
        String line = null;
        List service = new ArrayList();

        int i = 0;
        try {
                BufferedReader br = new BufferedReader(new FileReader(friServices));
                while ((line = br.readLine()) != null) {
                        System.out.println(line);
                        service.add(line);

                }

                Iterator it = service.iterator();
                String[] serviceList = new String[service.size()];
                while (it.hasNext()) {
                        serviceList[i] = (String) it.next();
                        i++;
                }
                return serviceList;
        } catch (FileNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }

        return null;

}

public void initialcurrentGroupFriendsInfor() {

        friendsList = Friends.list();

}

public String[] getCurrentGroupFriends() {
        initialcurrentGroupFriendsInfor();
        return friendsList;
}

/*
  * TODO:retrieval the existing group information, load into memory here,
  * just load the name of existing group
  */
private void initialGroupsInfor() {
        File groupList = new File(FilePath.HOME);
        // groupList.list();
        group = groupList.list();

}

/*
  * TODO:get group list
  */
public String[] getGroupList() {
        initialGroupsInfor();
        // System.out.println(group.toArray().toString());
        return group;
}
```

```
/*
 * storage adv in local system
 */
private void storeAdv(Advertisement adv) throws IOException {
        advPath.mkdirs();
        advF.createNewFile();
        Document adver = adv.getDocument(new MimeMediaType("text/xml"));
        FileOutputStream fos = new FileOutputStream(advF);
        adver.sendToStream(fos);
}

/*
 * initial specific group information from local adv
 */
private Advertisement retrievalGroupAdv(String groupName)
                throws IOException {
        FileInputStream fis = new FileInputStream(FilePath.HOME + groupName
                        + FilePath.ADV + "peerGroupAdv.xml");
        Advertisement adv = AdvertisementFactory.newAdvertisement(
                        new MimeMediaType("text/xml"), fis);
        return adv;
}

public void configPeer() {

}

private void configPipe() throws IOException {

        if (mygroup != null) {
                pipeService = mygroup.getPipeService();
                input = pipeService.createInputPipe(getMyJxtaPipeAdv(), this);
                // This pipe is a propagated pipe, therefore also bind to it
                output = pipeService.createOutputPipe(getMyJxtaPipeAdv(), 100);
        }

}

public void creatNewGroup(String name, String des) {

        ModuleImplAdvertisement TheModuleImplementationAdvertisement;

        // Creating a new peer group ID
        PeerGroupID ThePeerGroupID = IDFactory.newPeerGroupID();
        try {
            TheModuleImplementationAdvertisement = netPeerGroup
                    .getAllPurposePeerGroupImplAdvertisement();
            mygroup = netPeerGroup.newGroup(ThePeerGroupID,
                    TheModuleImplementationAdvertisement, name, des);
        } catch (PeerGroupException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (Exception ee) {

        }
        ;

        PeerGroupAdvertisement ThePeerGroupAdvertisement = mygroup
                .getPeerGroupAdvertisement();

        discovery.remotePublish(ThePeerGroupAdvertisement);
        System.out.println("created the group\n"
                + mygroup.getPeerGroupAdvertisement());
        currentGroupName = name;
        freshFilePath();
        if (monitorFile == true) {
            removeMonitorFile();
            addMonitorFile();
        } else
            addMonitorFile();

        try {
            storeAdv(ThePeerGroupAdvertisement);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public void creatGroupFromLocal(String groupName) throws Exception {

        PeerGroupAdvertisement ThePeerGroupAdvertisement = (PeerGroupAdvertisement)
AdvertisementFactory
                .newAdvertisement(PeerGroupAdvertisement.getAdvertisementType());

        Advertisement adv = retrievalGroupAdv(groupName);
        PeerGroupAdvertisement Ad = (PeerGroupAdvertisement) adv;
        ModuleImplAdvertisement groupImplAdv = netPeerGroup
                .getAllPurposePeerGroupImplAdvertisement();
        groupImplAdv.setModuleSpecID(Ad.getModuleSpecID());

        try {

            mygroup = netPeerGroup.newGroup(Ad.getPeerGroupID(), groupImplAdv,
                    groupName, Ad.getDescription());
        } catch (PeerGroupException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        // netPeerGroup.newGroup(adv., arg1, arg2, arg3)

        discovery.remotePublish(mygroup.getPeerGroupAdvertisement());

        currentGroupName = groupName;
        freshFilePath();
        if (monitorFile == true) {
            removeMonitorFile();
            addMonitorFile();
        } else
            addMonitorFile();
        System.out.println("created the group\n"
                + mygroup.getPeerGroupAdvertisement());

    }

    public void discoveryGroup(String name) throws InterruptedException {

        while (true) {
            if (joind) {
                // System.out.println("not get new group");

                discovery.removeDiscoveryListener(this);

                joind = false;
                return;
            }
            System.out.println("finding group");
            discovery.getRemoteAdvertisements(// no specific peer (propagate)
                    null, // Adv type
                    DiscoveryService.GROUP, // Attribute = any
```

```
                        "Name", // Value = any
                        name, // one advertisement response is all we are looking
                        // for
                        1, // no query specific listener. we are using a global
                        // listener
                        null);
                System.out.println("after finding group");

                try {

                    Thread.sleep(10000);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }

            }

        }

        /*
         * TODO: this method used to create local file system replica, each user
         * have its own sub-dictionary
         */
        public void createLocalCIRplica() {

            if (currentGroup.exists())
                System.out.println("Already have the group " + currentGroupName
                        + " folder");
            else {
                currentGroup.mkdirs();
                System.out.println("create the folderfor :" + currentGroupName
                        + "!");

            }

        }

        /*
         * TODO: this method used to create local user information replica each kind
         * of information represented by a xml file, which can store the information
         * you need to add in in future. The Friends is only Folder since it will
         * contains other user's information
         */
```

```
public void createLocalUserInfor(String userName) throws IOException {

	if (!(currentGroup.exists() && Name.exists())) {
		System.out.println("creating the user folders and files......");
		if (!currentGroup.exists())
			currentGroup.mkdirs();
		Name.mkdirs();
		NameF.createNewFile();
		Location.mkdir();
		LocationF.createNewFile();
		Status.mkdir();
		StatusF.createNewFile();
		Content.mkdir();
		ContentF.createNewFile();
		Service.mkdir();
		ServiceF.createNewFile();
		Templet.mkdir();
		TempletF.createNewFile();
		Friends.mkdir();
		System.out.println("finish create");

	} else {
		System.out.println("user:" + userName
				+ " have its home in local system!");
	}

}

/*
 * TODO:this class used for create friends folders for each CI
 */
public void createFriendsFolder() {
	File friends = new File("..\\CI\\" + currentGroupName + "\\Friends");
	friends.mkdir();
}

public String getUserInfor() {
	return peerName;
}

public void synchronizeUserInfor(String type, File file) {

}
```

```
/*
  * TODO: this method used to propagate local user information to remote
  * peers for synchronization
  *
  */
public void synchronizeUserInfor(String userName) throws IOException {
        File Location = new File("..\\CI\\" + currentGroupName + "\\"
                        + userName + "\\" + "Location");
        Location.mkdirs();
        File LocationFile = new File(Location, "test1.xml");
        if (!LocationFile.exists())
                LocationFile.createNewFile();

        File Status = new File("..\\CI\\" + currentGroupName + "\\" + userName
                        + "\\" + "Status");
        File Content = new File("..\\CI\\" + currentGroupName + "\\" + userName
                        + "\\" + "Content");
        File group = new File("..\\CI\\" + currentGroupName);
        if (group.exists())

                System.out.println("Already have the group folder");
        else {
                createLocalCIRplica();
                createLocalUserInfor(userName);
        }

}

public void sendFile(String Type, File file) throws IOException {
        // File file = new File(name);
        FileInputStream fis = new FileInputStream(file);
        Message msg = new Message();
        /*
          * */
        msg.addMessageElement(fileNameSpace, new StringMessageElement(
                        SenderName, peerName, null));
        msg.addMessageElement(fileNameSpace, new StringMessageElement(fileType,
                        Type, null));
        msg.addMessageElement(fileNameSpace, new InputStreamMessageElement(
                        "file", null, fis, null));
        if (msg != null)
                System.out.println("send file now");
        try {
                output.send(msg);
        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

public void sendMessage(String gram) {
        Message response = new Message();
        // The gram
        response.addMessageElement(messageNameSpace, new StringMessageElement(
                        SenderMessage, gram, null));
        // Our name
        response.addMessageElement(messageNameSpace, new StringMessageElement(
                        SenderName, peerName, null));
        try {
                // Send the message
                output.send(response);
        } catch (IOException io) {
                io.printStackTrace();
        }
}

private PipeAdvertisement getMyJxtaPipeAdv() {

        byte[] preCookedPID = { (byte) 0xD1, (byte) 0xD1, (byte) 0xD1,
                        (byte) 0xD1, (byte) 0xD1, (byte) 0xD1, (byte) 0xD1,
                        (byte) 0xD1, (byte) 0xD1, (byte) 0xD1, (byte) 0xD1,
                        (byte) 0xD1, (byte) 0xD1, (byte) 0xD1, (byte) 0xD1, (byte) 0xD1 };

        PipeID id = (PipeID) IDFactory.newPipeID(netPeerGroup.getPeerGroupID(),
                        preCookedPID);
        PipeAdvertisement pipeAdv = (PipeAdvertisement) AdvertisementFactory
                        .newAdvertisement(PipeAdvertisement.getAdvertisementType());
        pipeAdv.setPipeID(id);
        // the name really does not matter here, only for illustration
        pipeAdv.setName("test");
        pipeAdv.setType(PipeService.PropagateType);
        return pipeAdv;
}

private void joinGroup(Advertisement adv) {

        // The creation includes local publishing
        try {
                System.out.println("before join");
                mygroup = netPeerGroup.newGroup(adv);
                pipeService = mygroup.getPipeService();
                // Create the input pipe with this app as the message listener for
                // this
                // pipe
                input = pipeService.createInputPipe(getMyJxtaPipeAdv(), this);
                // This pipe is a propagated pipe, therefore also bind to it
                output = pipeService.createOutputPipe(getMyJxtaPipeAdv(), 100);
                currentGroupName = mygroup.getPeerGroupName();
                joind = true;
                System.out.println("after join");
        } catch (PeerGroupException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        } catch (IOException ee) {

        }
        System.out.println("we joined the group\n"
                        + mygroup.getPeerGroupAdvertisement());
        freshFilePath();
        System.out.println(mygroup.getPeerGroupAdvertisement());
        if (monitorFile == true) {
                removeMonitorFile();
                addMonitorFile();
        } else
                addMonitorFile();

        // long Duration = 1000 * 60 * 10;

        // Publishing the new group remotely
        discovery.remotePublish(adv);

        currentGroupName = mygroup.getPeerGroupName();
        return;

}

/**
 * Display messages as they arrive, if the message contains the string
 * "jxme" respond with a greeting
 */

public void pipeMsgEvent(PipeMsgEvent event) {
        Message msg = null;
        try {
            // grab the message from the event
            msg = event.getMessage();

            if (msg == null) {
                return;
            }
        } catch (Exception e) {
            e.printStackTrace();
            return;
        }
        System.out.println("get some message");

        ElementIterator ei = msg.getMessageElements();
        String name=null;
        String namespace=null;
        if (ei != null) {
            if (ei.hasNext()) {
                name=ei.next().toString();
                namespace=ei.getNamespace();
                System.out.println(name);
                System.out.println(namespace);
            }

        }else
            return;
//
        if (namespace.equals(fileNameSpace)&&!(name.equals(peerName))) {
            System.out.println("some file");
            String senderName = msg
                    .getMessageElement(fileNameSpace, SenderName).toString();
            System.out.println("sender by " + senderName);
            File friend = new File(Friends, senderName);
            File friendLocation = new File(friend, "Location");
            File friendStatus = new File(friend, "Status");
            File friendService = new File(friend, "Service");
            File friendContent = new File(friend, "Content");

            System.out.println("some file setp 2");
            if (!friend.exists()) {
                friend.mkdirs();
                friendLocation.mkdir();
                friendStatus.mkdir();
                friendService.mkdir();
                friendContent.mkdir();
            }
            MessageElement file = msg.getMessageElement(fileNameSpace, "file");
            if (file != null) {
                if (msg.getMessageElement(fileNameSpace, fileType).toString()
                        .equals(LocationFile)) {

                    File locationFile = new File(friendLocation, "location.xml");
                    writeFile(file, locationFile);

                }

                if (msg.getMessageElement(fileNameSpace, fileType).toString()
                        .equals(ServiceFile)) {

                    File serviceFile = new File(friendService, "service.xml");
                    writeFile(file, serviceFile);

                }
                if (msg.getMessageElement(fileNameSpace, fileType).toString()
                        .equals(StatusFile)) {

                    File statusFile = new File(friendLocation, "status.xml");
                    writeFile(file, statusFile);

                }
                if (msg.getMessageElement(fileNameSpace, fileType).toString()
                        .equals(TempletFile)) {

                    writeFile(file, TempletF);

                }

            }
            return;
        }

        if
(namespace.equals(messageNameSpace)&&!msg.getMessageElement(messageNameSpace,
SenderName)
                .toString().equals(peerName)) {
```

```
            System.out.println("some text message");
            String senderName = "unknown";

            // Get originator's name
            MessageElement nameEl = msg.getMessageElement(SenderName);
            if (nameEl != null) {
                senderName = nameEl.toString();
            }

            // now the message
            String senderMessage = null;
            MessageElement msgEl = msg.getMessageElement(SenderMessage);
            if (msgEl != null) {
                senderMessage = msgEl.toString();
            }

            // Get message
            if (senderMessage == null) {
                senderMessage = "[empty message]";
            }

            System.out.println(senderName + "> " + senderMessage);
            // return;
        }

        return;
    }

    private void writeFile(MessageElement fileElement, File targetFile) {

        if (!targetFile.exists()) {
            try {
                targetFile.createNewFile();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        try {
            os = new FileOutputStream(targetFile);
            fileElement.sendToStream(os);
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
```

```
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	public void discoveryEvent(DiscoveryEvent e) {
		System.out.println("got it!!");
		DiscoveryResponseMsg res = e.getResponse();

		Advertisement adv;
		Enumeration en = res.getAdvertisements();

		if (en != null) {
			while (en.hasMoreElements()) {
				adv = (Advertisement) en.nextElement();
				if (adv.getAdvType() == "jxta:PGA") {

					if (!joind) {
						// find=true;
						System.out.println(" [    Got a Discovery Response ["
								+ res.getResponseCount()
								+ " elements]    from peer : " + e.getSource()
								+ "    ]");
						System.out.println(adv);
						System.out.println(adv.getAdvType());

						System.out.println("now join the    found group!!!!");

						joinGroup(adv);
					}

				}

			}
		}

	}

	private void addMonitorFile() {
		// monitor = new FileMonitor(1000);
		monitor.addFile(NameF);
		monitor.addFile(LocationF);
		monitor.addFile(StatusF);
```

```
        monitor.addFile(ContentF);
        monitor.addFile(ServiceF);
        monitor.addFile(TempletF);
        monitor.addFile(NameF);
        monitor.addFile(new File(FilePath.HOME));
        // monitor.addFile(new File(Friends,"/wang/Service/service.xml"));
        monitor.addFile(new File(Friends + "/"));
        if (new File(Friends + "/").list() != null) {
                String[] FriendsList = new File(Friends + "/").list();
                int i = FriendsList.length;
                for (int j = 0; j < i; j++) {
                        monitor.addFile(new File(Friends, "/" + FriendsList[j]
                                        + "/Service/service.xml"));
                }

        }
        monitorFile = true;

}

private void removeMonitorFile() {
        // monitor = new FileMonitor(1000);
        monitor.removeFile(NameF);
        monitor.removeFile(LocationF);
        monitor.removeFile(StatusF);
        monitor.removeFile(ContentF);
        monitor.removeFile(ServiceF);
        monitor.removeFile(TempletF);
        monitor.removeFile(NameF);
        monitor.removeFile(new File(FilePath.HOME));
        // monitor.removeFile(new File(Friends,"/wang/Service/service.xml"));
        monitor.removeFile(new File(Friends + "/"));
        if (new File(Friends + "/").list() != null) {
                String[] FriendsList = new File(Friends + "/").list();
                int i = FriendsList.length;
                for (int j = 0; j < i; j++) {
                        monitor.removeFile(new File(Friends, "/" + FriendsList[j]
                                        + "/Service/service.xml"));
                }

        }
        monitorFile = false;

}
```

```
        public void reMonitor() {
                removeMonitorFile();
                addMonitorFile();
        }

}
```

Class FilePath

```
package service;

public class FilePath {
    public final static String HOME = "..\\CI\\";
    public final static String ADV ="\\adv\\";
    public final static String NAME = "Myfolder\\Name\\";
    public final static String CONTENT = "Myfolder\\Content\\";
    public final static String STATUS = "Myfolder\\Status\\";
    public final static String SERVICE = "Myfolder\\Service\\";
    public final static String FRIENDS = "Myfolder\\Friends\\";
    public final static String LOCATION = "Myfolder\\Location\\";
    public final static String TEMPLET = "Myfolder\\Templet\\";

}
```

Package fileMonitor

Interface FileListener

```
/*
 * This code is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This code is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this program; if not, write to the Free
 * Software Foundation, Inc., 59 Temple Place - Suite 330, Boston,
  * MA    02111-1307, USA.
  */
package fileMonitor;


import java.io.File;



/**
  * Interface for listening to disk file changes.
  * @see FileMonitor
  *
  * @author <a href="mailto:jacob.dreyer@geosoft.no">Jacob Dreyer</a>
  */
public interface FileListener
{
    /**
      * Called when one of the monitored files are created, deleted
      * or modified.
      *
      * @param file    File which has been changed.
      */
    void fileChanged (File file);
}
```

Class FileMonitor

```
/*
  * This code is free software; you can redistribute it and/or
  * modify it under the terms of the GNU Lesser General Public
  * License as published by the Free Software Foundation; either
  * version 2.1 of the License, or (at your option) any later version.
  *
  * This code is distributed in the hope that it will be useful,
  * but WITHOUT ANY WARRANTY; without even the implied warranty of
  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
  * GNU Lesser General Public License for more details.
  *
  * You should have received a copy of the GNU Lesser General Public
  * License along with this program; if not, write to the Free
  * Software Foundation, Inc., 59 Temple Place - Suite 330, Boston,
  * MA    02111-1307, USA.
  */
```

```
package fileMonitor;



import java.util.*;
import java.io.File;
import java.lang.ref.WeakReference;



/**
 * Class for monitoring changes in disk files.
 * Usage:
 *
 *      1. Implement the FileListener interface.
 *      2. Create a FileMonitor instance.
 *      3. Add the file(s)/directory(ies) to listen for.
 *
 * fileChanged() will be called when a monitored file is created,
 * deleted or its modified time changes.
 *
 * @author <a href="mailto:jacob.dreyer@geosoft.no">Jacob Dreyer</a>
 */
public class FileMonitor
{
  private Timer        timer_;
  private HashMap      files_;       // File -> Long
  private Collection   listeners_;   // of WeakReference(FileListener)


  /**
   * Create a file monitor instance with specified polling interval.
   *
   * @param pollingInterval   Polling interval in milli seconds.
   */
  public FileMonitor (long pollingInterval)
  {
    files_     = new HashMap();
    listeners_ = new ArrayList();

    timer_ = new Timer (true);
    timer_.schedule (new FileMonitorNotifier(), 0, pollingInterval);
  }
```

```
/**
  * Stop the file monitor polling.
  */
public void stop()
{
    timer_.cancel();
}


/**
  * Add file to listen for. File may be any java.io.File (including a
  * directory) and may well be a non-existing file in the case where the
  * creating of the file is to be trepped.
  * <p>
  * More than one file can be listened for. When the specified file is
  * created, modified or deleted, listeners are notified.
  *
  * @param file    File to listen for.
  */
public void addFile (File file)
{
    if (!files_.containsKey (file)) {
        long modifiedTime = file.exists() ? file.lastModified() : -1;
        files_.put (file, new Long (modifiedTime));
    }
}



/**
  * Remove specified file for listening.
  *
  * @param file    File to remove.
  */
public void removeFile (File file)
{
    files_.remove (file);
}



/**
```

```
  * Add listener to this file monitor.
  *
  * @param fileListener    Listener to add.
  */
public void addListener (FileListener fileListener)
{
    // Don't add if its already there
    for (Iterator i = listeners_.iterator(); i.hasNext(); ) {
        WeakReference reference = (WeakReference) i.next();
        FileListener listener = (FileListener) reference.get();
        if (listener == fileListener)
            return;
    }

    // Use WeakReference to avoid memory leak if this becomes the
    // sole reference to the object.
    listeners_.add (new WeakReference (fileListener));
}



/**
  * Remove listener from this file monitor.
  *
  * @param fileListener    Listener to remove.
  */
public void removeListener (FileListener fileListener)
{
    for (Iterator i = listeners_.iterator(); i.hasNext(); ) {
        WeakReference reference = (WeakReference) i.next();
        FileListener listener = (FileListener) reference.get();
        if (listener == fileListener) {
            i.remove();
            break;
        }
    }
}



/**
  * This is the timer thread which is executed every n milliseconds
  * according to the setting of the file monitor. It investigates the
  * file in question and notify listeners if changed.
```

```
    */
   private class FileMonitorNotifier extends TimerTask
   {
      public void run()
      {
         // Loop over the registered files and see which have changed.
         // Use a copy of the list in case listener wants to alter the
         // list within its fileChanged method.
         Collection files = new ArrayList (files_.keySet());

         for (Iterator i = files.iterator(); i.hasNext(); ) {
            File file = (File) i.next();
            long lastModifiedTime = ((Long) files_.get (file)).longValue();
            long newModifiedTime    = file.exists() ? file.lastModified() : -1;

            // Chek if file has changed
            if (newModifiedTime != lastModifiedTime) {

               // Register new modified time
               files_.put (file, new Long (newModifiedTime));

               // Notify listeners
               for (Iterator j = listeners_.iterator(); j.hasNext(); ) {
                  WeakReference reference = (WeakReference) j.next();
                  FileListener listener = (FileListener) reference.get();

                  // Remove from list if the back-end object has been GC'd
                  if (listener == null)
                     j.remove();
                  else
                     listener.fileChanged (file);
               }
            }
         }
      }
   }



}
```

Package CIM

Class CIM

```
package CIM;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

import fileMonitor.FileListener;

import service.jxmeUbiService;
import tool.CIMTool;

public class CIM {
        private static final String LocationFile = "Location";
        private static final String StatusFile = "Status";
        private static final String ServiceFile = "Service";
        private static final String ContentFile = "Content";
        private static final String TempletFile = "Templet";
        private static InputStreamReader cin = new InputStreamReader(System.in);
        private static OutputStreamWriter osw = new OutputStreamWriter(System.out);
        private static BufferedReader readrer = new BufferedReader(cin);
        private static jxmeUbiService jus = null;
        private String userName = null;

        private CIMTool tool;

        public void initialJxta(String name, int port) {
                jus = new jxmeUbiService();
                try {
                        try {
                                jus.startJxta(name, port);
                                jus.startService();

                        } catch (InterruptedException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }

                        userName = name;
                        // jus.creatNewGroup("want", "test2");
                        // jus.discoveryGroup("ubicollab");
                        // jus.createLocalCIRplica();
```

```
            // jus.createLocalUserInfor("xiaobo1");
            // jus.synchronizeUserInfor("xiaobo");
            // File file = jus.ServiceF;
            // System.out.println(file);
            // jus.sendFile("Service", file);

            // jus.discoveryGroup("xiaobo test group");
            // jus.createLocalCIRplica();
            // jus.createLocalUserInfor("xiaobo");

            // jus.discoveryGroup("xiaobo test group");
            // chatDemo.discoveryGroup("xiaobo test group");
            // chatDemo.printMemStat();

            // chatDemo.discoveryGroup("xiaobo test group");

            /*
             * while (true) { String message = readrer.readLine();
             * jus.sendMessage(message); }
             *
             */

        } catch (IOException io) {
            io.printStackTrace();
        }

        /*
         * synchronized(jus) { //The intention with the wait is to ensure the
         * app continues to run //run. try { jus.wait(); } catch
         * (InterruptedException ie) { Thread.interrupted(); } }
         *
         */

    }

    public void setCIMTool(CIMTool tool) {
        jus.tool = tool;
        this.tool = tool;
    }

    public void addFileListener() {

        jus.addListener(tool);
    }
```

```
public void synchronizeFile(String type) {
        if (type.equals(LocationFile))
                try {
                        jus.sendFile(type, jus.LocationF);
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

        if (type.equals(StatusFile))
                try {
                        jus.sendFile(type, jus.StatusF);
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

        if (type.equals(ServiceFile))
                try {
                        jus.sendFile(type, jus.ServiceF);
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

        if (type.equals(ContentFile))
                try {
                        jus.sendFile(type, jus.ContentF);
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

        if (type.equals(TempletFile))
                try {
                        jus.sendFile(type, jus.TempletF);
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
}

public String[] getGroupInfor() {
        // System.out.println((String[])jus.getGroupList().toArray());
        return jus.getGroupList();
}

public void createNewGroup(String name, String des) {
        jus.creatNewGroup(name, des);
        jus.createLocalCIRplica();
        try {
                jus.createLocalUserInfor(userName);
        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

public void startLocalGroup(String name) {
        try {
                jus.creatGroupFromLocal(name);
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

public void discoveryCI(String name) {
        try {
                jus.discoveryGroup(name);
        } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

public String[] getFriendsList() {
        return jus.getCurrentGroupFriends();
}

public String[] getFriendServices(String name) {
        return jus.getFriendServices(name);
}

public void remonitor() {
        jus.reMonitor();
}
```

```
        public File getFile(String type) {
                if (type.equals(LocationFile))
                        return jus.LocationF;

                if (type.equals(StatusFile))
                        return jus.StatusF;

                if (type.equals(ServiceFile))
                        return jus.ServiceF;

                if (type.equals(ContentFile))
                        return jus.ContentF;

                if (type.equals(TempletFile))
                        return jus.TempletF;
                return null;
        }

        public void sendMessage(String message){
                jus.sendMessage(message);
        }
}
```

Package CI

I don¡t include the classes in this package since most of them are abstracted and have not real used in CIM research at this moment.

Package CIMTool

Class CIMTool

```
package tool;

import java.io.BufferedReader;

public class CIMTool implements FileListener {

    protected Shell shlUbibuddy;
    private Text text;
    private CIM sm;
    Combo groups;
    private Text text_1;
private Label status;
static CIMTool window;
private Display display;
private TreeItem trtmMyFriends;
private Tree tree;
private Button btnChangeStatus;
private Button btnNewWindow;
private Shell child;

// static Thread t;

/**
 * Launch the application.
 */

public static void start() {

    try {
        window = new CIMTool();
        window.open();

    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void fileChanged(File file) {
    System.out.println("changed");

    if (file.getName().equals("service.xml")) {
        File Service = new File(file.getParent());
        String name = new File(Service.getParent()).getName();
        updateService(name);
    } else if (file.getName().equals("Friends")) {
        updateFriends();
    } else if (file.getName().equals("status.xml")) {

        try {
            String line = null;
            BufferedReader br = new BufferedReader(new FileReader(sm
                    .getFile("Status")));
            if ((line = br.readLine()) != null) {
                System.out.println("my status " + line);
                updateStatus(line);
            }
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

}

/**
 * Open the window.
 */
public void open() {
    display = Display.getDefault();
    createContents();

    shlUbibuddy.open();
    shlUbibuddy.layout();
    while (!shlUbibuddy.isDisposed()) {
        if (!display.readAndDispatch()) {
            display.sleep();
        }
    }

}

/**
 * Create contents of the window.
 */
protected void createContents() {
    shlUbibuddy = new Shell(SWT.SHELL_TRIM);
    shlUbibuddy.setLayout(new GridLayout(2, false));
    shlUbibuddy.setSize(450, 300);
    shlUbibuddy.setText("UbiBuddy");
    {
        Button btnStartUbicollab = new Button(shlUbibuddy, SWT.NONE);
        {
            GridData gridData = new GridData(GridData.BEGINNING,
                    GridData.CENTER, true, false, 1, 1);
            btnStartUbicollab.setLayoutData(gridData);
        }
        btnStartUbicollab.addMouseListener(new MouseListener() {
            public void mouseDoubleClick(MouseEvent arg0) {
            }

            public void mouseDown(MouseEvent arg0) {
                sm = new CIM();
                sm.initialJxta("xiaoboo", 9910);
                sm.setCIMTool(window);
                sm.addFileListener();
                System.out.println("added listener");
                // sm.addFileListener();
                String[] l = sm.getGroupInfor();
                groups.setItems(l);

            }

            public void mouseUp(MouseEvent arg0) {
            }
        });
        btnStartUbicollab.setText("Start Ubicollab");
    }
    new Label(shlUbibuddy, SWT.NONE);
    {
        text = new Text(shlUbibuddy, SWT.BORDER);
        {
            GridData gridData = new GridData(GridData.BEGINNING,
                    GridData.CENTER, true, false, 1, 1);
            text.setLayoutData(gridData);
        }
        text.addVerifyListener(new VerifyListener() {
            public void verifyText(VerifyEvent arg0) {
            }
        });
        text.addModifyListener(new ModifyListener() {
            public void modifyText(ModifyEvent arg0) {
            }
        });
    }
    {
        Button btnCi = new Button(shlUbibuddy, SWT.NONE);
        {
            GridData gridData = new GridData(GridData.BEGINNING,
                    GridData.CENTER, true, false, 1, 1);
            btnCi.setLayoutData(gridData);
        }
        btnCi.addMouseListener(new MouseListener() {
            public void mouseDoubleClick(MouseEvent arg0) {
            }

            public void mouseDown(MouseEvent arg0) {
                sm.createNewGroup(text.getText(), "test");
                groups.add(text.getText());
                // text.getText();
                // text.setTopIndex(0);
                // text.setSelection(0, 10);
                // text.insert("hello world");
                // System.out.println("push me");
            }

            public void mouseUp(MouseEvent arg0) {
            }
        });
        btnCi.addKeyListener(new KeyListener() {
            public void keyPressed(KeyEvent arg0) {
            }

            public void keyReleased(KeyEvent arg0) {
            }
        });
        btnCi.setText("createNewCI");
    }
    {
        groups = new Combo(shlUbibuddy, SWT.NONE);
        {
            GridData gridData = new GridData(GridData.FILL,
                    GridData.CENTER, true, false, 1, 1);
            groups.setLayoutData(gridData);
        }
        groups.setText("CIS");

    }

    Button btnStartlocalci = new Button(shlUbibuddy, SWT.NONE);
    btnStartlocalci.addMouseListener(new MouseListener() {
        public void mouseDoubleClick(MouseEvent arg0) {
        }

        public void mouseDown(MouseEvent arg0) {
                sm.startLocalGroup(groups.getText());
                String[] friends = sm.getFriendsList();
                if (friends.length != 0) {
                    trtmMyFriends.dispose();
                    trtmMyFriends = new TreeItem(tree, SWT.NONE);
                    trtmMyFriends.setText(groups.getText());
                    trtmMyFriends.setExpanded(true);

                    for (int i = 0; i < friends.length; i++) {
                        TreeItem name = new TreeItem(trtmMyFriends,
SWT.NONE);
                        name.setText(friends[i]);
                        name.setExpanded(true);
                        String[] Services =
sm.getFriendServices(friends[i]);
                        for (int j = 0; j < Services.length; j++) {
                            TreeItem service = new TreeItem(name,
SWT.NONE);
                            service.setText(Services[j]);
                        }
                    }
                }

            }

            public void mouseUp(MouseEvent arg0) {
            }
        });
        {
            GridData gridData = new GridData(GridData.FILL,
GridData.CENTER,
                    true, false, 1, 1);
            btnStartlocalci.setLayoutData(gridData);
        }
        btnStartlocalci.setText("startLocalCI");
        new Label(shlUbibuddy, SWT.NONE);
        new Label(shlUbibuddy, SWT.NONE);

        final Combo fileList = new Combo(shlUbibuddy, SWT.NONE);
        fileList.addSelectionListener(new SelectionListener() {
            public void widgetDefaultSelected(SelectionEvent arg0) {
            }

            public void widgetSelected(SelectionEvent arg0) {
            }
        });
        fileList.setText("choice");
        fileList.setItems(new String[] { "Location", "Status", "Service",
                "Content", "Templet" });
        {
            GridData gridData = new GridData(GridData.FILL,
GridData.CENTER,
                    true, false, 1, 1);
            fileList.setLayoutData(gridData);
        }

        {
            Button btnSynchronize = new Button(shlUbibuddy, SWT.NONE);
            {
                GridData gridData = new GridData(GridData.FILL,
                        GridData.CENTER, true, false, 1, 1);
                btnSynchronize.setLayoutData(gridData);
            }
            btnSynchronize.addMouseListener(new MouseListener() {
                public void mouseDoubleClick(MouseEvent arg0) {
                }

                public void mouseDown(MouseEvent arg0) {
                    sm.synchronizeFile(fileList.getText());
                    // System.out.println(fileList.getText());
                }

                public void mouseUp(MouseEvent arg0) {
                }
            });
            btnSynchronize.setText("synchronize");
        }
        new Label(shlUbibuddy, SWT.NONE);
        new Label(shlUbibuddy, SWT.NONE);

        text_1 = new Text(shlUbibuddy, SWT.BORDER);
        {
            GridData gridData = new GridData(GridData.FILL,
GridData.CENTER,
                    true, false, 1, 1);
            text_1.setLayoutData(gridData);
        }
        Button btnDiscoveryci = new Button(shlUbibuddy, SWT.NONE);
        btnDiscoveryci.addMouseListener(new MouseListener() {
            public void mouseDoubleClick(MouseEvent arg0) {
            }

            public void mouseDown(MouseEvent arg0) {
                sm.discoveryCI(text_1.getText());
                groups.add(text_1.getText());
            }

            public void mouseUp(MouseEvent arg0) {
            }
        });
        {
            GridData gridData = new GridData(GridData.FILL,
GridData.CENTER,
                    true, false, 1, 1);
            btnDiscoveryci.setLayoutData(gridData);
        }
        btnDiscoveryci.setText("discoveryCI");

        status = new Label(shlUbibuddy, SWT.NONE);
        {
            GridData gridData = new GridData(GridData.BEGINNING,
                    GridData.CENTER, true, false, 1, 1);
            status.setLayoutData(gridData);
        }
        status.setText("offline");

        btnChangeStatus = new Button(shlUbibuddy, SWT.NONE);
        {
            GridData gridData = new GridData(GridData.BEGINNING,
                    GridData.CENTER, true, false, 1, 1);
            btnChangeStatus.setLayoutData(gridData);
        }
        btnChangeStatus.addMouseListener(new MouseListener() {
            public void mouseDoubleClick(MouseEvent arg0) {
            }

            public void mouseDown(MouseEvent arg0) {

                if (status.getText().equals("online"))
                    writeToFile(sm.getFile("Status"), "offline");
                else
                writeToFile(sm.getFile("Status"), "online");
            }

            public void mouseUp(MouseEvent arg0) {
            }
        });
        btnChangeStatus.setText("Change status");

        tree = new Tree(shlUbibuddy, SWT.BORDER);
        {
            GridData gridData = new GridData(GridData.FILL,
GridData.FILL,
                    true, true, 1, 1);
            tree.setLayoutData(gridData);
        }

        trtmMyFriends = new TreeItem(tree, SWT.NONE);
        trtmMyFriends.setText("My Friends");
        trtmMyFriends.setExpanded(true);

        btnNewWindow = new Button(shlUbibuddy, SWT.NONE);
        {
            GridData gridData = new GridData(GridData.BEGINNING,
                    GridData.CENTER, true, false, 1, 1);
            btnNewWindow.setLayoutData(gridData);
        }
        btnNewWindow.addMouseListener(new MouseListener() {
            public void mouseDoubleClick(MouseEvent arg0) {
            }

            public void mouseDown(MouseEvent arg0) {
                InputDialog dlg = new InputDialog(Display.getCurrent()
                        .getShells()[0]);
                String input = dlg.open();

                /*
                 * Shell shell = new Shell(); shell.setLayout(new
FormLayout());
                 * shell.setVisible(true); shell.setText("Select Column
Data");
                 *
                 *
                 * Label label = new Label(shell, SWT.NONE);
                 * label.setText("hihi"); GridData data = new GridData();
                 * data.horizontalSpan = 2; label.setLayoutData(data);
                 *
                 *
                 *
                 * Label label1 = new Label(shell, SWT.NONE);
                 * label1.setText("YOUPI"); FormData formData = new
FormData();
                 * formData.bottom = new FormAttachment(100, 0);
formData.top =
                 * new FormAttachment(0, 0); formData.left = new
                 * FormAttachment(0, 0); formData.right = new
                 * FormAttachment(100, 0);
label1.setLayoutData(formData);
                 * shell.pack(); shell.setLocation(200, 200);
shell.setSize(300,
                 * 100); shell.open();
                 */

            }

            public void mouseUp(MouseEvent arg0) {
            }
        });
        btnNewWindow.setText("new window");

    }

    protected void updateStatus(String s) {
        System.out.println("file changed");
        sm.synchronizeFile("Status");
        final String x = s;
        if (this.status.isDisposed())
            return;

        display.asyncExec(new Runnable() {
            public void run() {
                status.setText(x);
                status.redraw();
            }
        });

        System.out.println("file changed");

    }
```

```
protected void updateService(String name) {

    System.out.println("file changed");
    sm.synchronizeFile("Service");
    final String x = name;
    if (this.status.isDisposed())
        return;

    display.asyncExec(new Runnable() {
        public void run() {
            String[] friends = sm.getFriendsList();

            trtmMyFriends.dispose();
            trtmMyFriends = new TreeItem(tree, SWT.NONE);
            trtmMyFriends.setText(groups.getText());
            trtmMyFriends.setExpanded(true);

            for (int i = 0; i < friends.length; i++) {
                TreeItem name = new TreeItem(trtmMyFriends, SWT.NONE);
                name.setText(friends[i]);
                name.setExpanded(true);
                String Services[] = sm.getFriendServices(friends[i]);
                for (int j = 0; j < Services.length; j++) {
                    TreeItem service = new TreeItem(name, SWT.NONE);
                    service.setText(Services[j]);

                }
            }

        }
    });

    System.out.println("file changed");

}

protected void updateFriends() {
    System.out.println("file changed");

    if (this.status.isDisposed())
        return;
    sm.remonitor();
    display.asyncExec(new Runnable() {
            public void run() {
                String[] friends = sm.getFriendsList();

                trtmMyFriends.dispose();
                trtmMyFriends = new TreeItem(tree, SWT.NONE);
                trtmMyFriends.setText(groups.getText());
                trtmMyFriends.setExpanded(true);

                for (int i = 0; i < friends.length; i++) {
                    TreeItem name = new TreeItem(trtmMyFriends, SWT.NONE);
                    name.setText(friends[i]);
                    name.setExpanded(true);
                    String Services[] = sm.getFriendServices(friends[i]);
                    for (int j = 0; j < Services.length; j++) {
                        TreeItem service = new TreeItem(name, SWT.NONE);
                        service.setText(Services[j]);
                    }
                }

            }
        });

        System.out.println("file changed");

    }

    private void writeToFile(File file, String content) {
        try {
            // Create file
            FileWriter fstream = new FileWriter(file);
            BufferedWriter out = new BufferedWriter(fstream);
            out.write(content);
            // Close the output stream
            out.close();
        } catch (Exception e) {// Catch exception if any
            System.err.println("Error: " + e.getMessage());
        }
    }

    class InputDialog extends Dialog {

        private String message;

        private String input;
```

```
// Shell shell;

public InputDialog(Shell parent) {

    this(parent, SWT.DIALOG_TRIM | SWT.APPLICATION_MODAL);
}

public InputDialog(Shell parent, int style) {
    super(parent, style);
    setText("Chating with group Dialog");
    setMessage("Please enter chatting content");
}

public String getMessage() {
    return message;
}

public void setMessage(String message) {
    this.message = message;
}

public String getInput() {
    return input;
}

public void setInput(String input) {
    this.input = input;
}

public String open() {
    // Display d= new Display();

    child = new Shell(getParent(), SWT.DIALOG_TRIM);
    child.setText(getText());
    child.forceActive();
    createContents(child);


    child.pack();
    child.open();

    // display.dispose();
```

```
        return input;
    }

    private void createContents(final Shell shell1) {

        shell1.setLayout(new GridLayout(5, true));
        shell1.forceActive();
        shell1.setVisible(true);
        shell1.setSize(450, 300);

        Label label = new Label(shell1, SWT.NONE);
        label.setText(message);
        GridData data = new GridData();
        data.horizontalSpan = 5;
        data.verticalSpan = 3;
        label.setLayoutData(data);

        final Text send = new Text(shell1, SWT.BORDER);
        data = new GridData(GridData.FILL_HORIZONTAL);
        data.horizontalSpan = 5;
        send.setLayoutData(data);



        Button ok = new Button(shell1, SWT.PUSH);
        ok.setText("Sending");
        data = new GridData(GridData.FILL_HORIZONTAL);
        data.horizontalSpan = 2;
        ok.setLayoutData(data);
        ok.addListener(SWT.Selection, new Listener() {
            public void handleEvent(Event e) {
                sm.sendMessage(send.getText());
            }
        });

        Button cancel = new Button(shell1, SWT.PUSH);
        cancel.setText("Cancel");
        data = new GridData(GridData.FILL_HORIZONTAL);
        data.horizontalSpan = 2;
        cancel.setLayoutData(data);

        final Text receive = new Text(shell1, SWT.BORDER);
        data = new GridData(GridData.FILL_HORIZONTAL);
        data.horizontalSpan = 4;
        send.setLayoutData(data);
```

```
            shell1.setDefaultButton(ok);

        }
    }
}
```

Package osgi

Class Activator

```
/*
 * Created on Fri May 09 18:34:11 CST 2008
 */
package osgi;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

import tool.CIMTool;

public class Activator implements BundleActivator {

   /* (non-Javadoc)
    * @see org.osgi.framework.BundleActivator#start(org.osgi.framework.BundleContext)
    */

   public void start(BundleContext context) throws Exception {
           CIMTool.start();
   }

   /* (non-Javadoc)
    * @see org.osgi.framework.BundleActivator#stop(org.osgi.framework.BundleContext)
    */
   public void stop(BundleContext context) throws Exception {
   }
}
```

//

Source code can be found in disk also

A 3 Deployment

The details of deployment can be found in the readme.txt in the attached disk.

Reference

1. **Farshchian, B.A. and M. Divitini, *UbiCollab Architecture White Paper.* IDI Technical Report, 2007.**
2. **ANDY CRABTREE and T.R.S.B., *Moving with the Times: IT Research and the Boundaries of CSCW.* 2005. p. 217-251.**
3. **Paul Luff, C.H., *Mobility in Collaboration*, in *ACM conference on Computer supported cooperative work.* 1998, ACM: Washington, United States.**
4. **Abowd, G.D. and E.D. Mynatt, *Charting Past, Present, and Future Research in Ubiquitous Computing.* ACM Transactions on Computer-Human Interaction (TOCHI), 2000: p. 29-58.**
5. **Gong, L., *JXTA: a network programming environment.* Internet Computing, IEEE,, 2001. 5(3): p. 88-95.**
6. **Greenberg, S. and D. Marwood. *Real time groupware as a distributed system: concurrency control and its effect on the interface*. in *Proceedings of the 1994 ACM conference on Computer supported cooperative work.* 1994.**
7. **Carstensen, P. and K. Schmidt, *Computer Supported Cooperative Work: New Challenges to System Design.* Handbook of Human Factors, ed. K.I. (ed.). 2002.**
8. **Dey, A.K., C. Edwards, and D. Salber, *A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications.* Human-Computer Interaction, 2001. 16: p. 97-166.**
9. **Jones, Q. and S.A. Grandhi, *P3 systems: putting the place back into social networks.* Internet Computing, IEEE, 2005. 9(5): p. 38-46.**
10. **Coulouris, J.D. and T. Kindberg, eds. *Distributed Systems: Concepts and Design.* 2005, Addison Wesley.**
11. **Weiser, M., *The computer for the 21st century.* sci, am, 1991(265(3)): p. 94-104.**
12. **Ballesteros, F.J., et al., *Plan B: Using Files instead of Middleware Abstractions.* Pervasive Computing, IEEE, 2007. 6(3): p. 58 - 65.**
13. **Bhana, I. and D. Johnson, *A Peer-to-Peer Approach to Content Dissemination and Search in Collaborative Networks.* Computational Science: Lecture Notes in Computer Science 2005. 3516/2005: p. 391-398.**
14. **Ripeanu, M. *Peer-to-Peer Architecture Case Study: Gnutella Network*. in *First International Conference on Peer-to-Peer Computing (P2P'01)*. 2001.**
15. **Lv, Q., et al. *Search and replication in unstructured peer-to-peer networks.* in *16th international conference on Supercomputing.* 2002. New York: ACM.**
16. **Androutsellis-Theotokis, S. and D. Spinellis, *A survey of peer-to-peer content distribution technologies.* ACM Computing Surveys (CSUR), 2004. 36(4): p. 335-371.**
17. **Oster, G., et al. *Data consistency for P2P collaborative editing.* in *anniversary conference on Computer supported cooperative work*. 2006: ACM.**

18. **Sun, C., et al.,** ***Achieving Convergence, Causality Preservation, and Intention Preservation in Real-Time Cooperative Editing Systems.*** **ACM Transactions on Computer-Human Interaction, 1998. 5(1): p. 63-108.**
19. **Kindberg, T. and A. Fox,** ***System software for ubiquitous computing.*** **Pervasive Computing, IEEE, 2002. 1(1)(70-81).**
20. **Satyanarayanan, M.,** ***Pervasive computing: vision and challenges.*** **Personal Communications, IEEE, 2001. 8(4): p. 10-17.**
21. **ErichGamma, et al.,** ***Design Patterns: Elements of Reusable Object-Oriented Software.*** **1994: AddisonWesley.**