INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Context Based Delivery of Voice Messages

Yongseok Choi

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

April 2002

ABSTRACT

Context Based Delivery of Voice Messages

Yongseok Choi

As a new technology for communication enriches, users are often overwhelmed not only by the technology itself but also by the information that comes with it. Especially, message receivers become more vulnerable to interruption of their normal workflow than the past, as there has been such an increase in the number and volume of messages received. People need to keep their attention focused on their mainline task by minimizing interruptions from unnecessary messages. One possible solution to tackle this problem is to give users a personalized control of messages according to their current contexts.

The purpose of this study is to develop an agent-based architecture based on the blackboard communication model that captures a user's contexts to enhance a user's control over the incoming messages. Some of the strengths of using the blackboard communication model are the ability to separate the concerns among the agents and to allow for the replacement of one agent with another agent that might be more effective in doing the intended functionality. In the proposed system architecture, the core parts are: characterizing and representing the user's context, allowing the user to set rules of preference dynamically, and managing the delivery of incoming messages. As a result, this system enables users to receive messages according to the paradigm, 'where you are', 'what you are doing', 'when a message arrives', 'who is calling', and what you want to do with the incoming message in the current context.

Acknowledgements

I am very grateful to my supervisor, Dr. T. Radhakrishnan, for directing me and providing advice. His suggestions, ideas and feedbacks were the root of this thesis.

I also want to thank Tom Gray and Ramiro Liscano at the MITEL Networks for being generous with their time and sharing experience throughout the project. All the Strategic Technology Group members in the MITEL Networks deserve my thanks as well.

My special thank goes to Jennifer Scott for her time and effort to proofread every chapter of my thesis.

To my mom and dad, thanks for your endless love and support.

Insook, my wife, thanks for always being with me. Nothing can be done without you.

Contents

List of Figures viii

1 Introduction **1**

1.1 Research Focus 2

1.2 Thesis Overview 3

2 Issues Related to "Context" **4**

2.1 Overview 4

2.2 Characterization of Context 4

2.2.1 Context Definition 4

2.2.2 Location 6

2.2.3 Activity 10

2.2.4 Time 11

2.3 Relationship Between Message Sender and Receiver 12

2.4 User Preference Rule 13

2.5 User Preference Rule Conflict 14

2.5.1 Detecting Conflicts 15

2.5.2 Resolving Conflicts 16

2.6 Associated Issues 17

2.6.1 Message Delivery Classification 17

2.6.2 Context Model and Representation 18

2.7 Summary 21

3 A System for Context Based Message Delivery **22**

3.1 System Overview 22

3.2 Architecture 26

3.2.1 User Interface 26

3.2.2 Event Handler 26

3.2.3 Application Level 27
3.2.4 Topology 31
3.3 Knowledge Categories 32
3.3.1 User Information 32
3.3.2 Call Information 34
3.4 Module Interactions 36
3.4.1 Use Case Diagram 36
3.4.2 State Chart Diagram 37
3.4.3 Call Processing 38

4 Implementation and Testing of ACD System 40
4.1 Server 41
4.1.1 Server Login 42
4.1.2 Server Main Window 43
4.1.2.1 System Knowledge Management 43
4.1.2.2 Startup Agents 45
4.1.2.3 Call Simulation 49
4.2 Client 50
4.2.1 Client Login 51
4.2.2 User Registration 52
4.2.3 Client Main Window 53
4.2.3.1 User Information 53
4.2.3.2 Knowledge Management 54
4.2.4 Client Agents 64

5 Related Research and Systems Dealing with Context 67
5.1 Active Badge System 67
5.2 Locust Swarm 70
5.3 Hanging Messages 71
5.4 Other Related Research 73
5.5 Commercial Systems 73
5.6 Summary 75

6 Concluding Remarks **76**
6.1 Contributions 76
6.2 Future Research and Implementation 78
6.2.1 Implement with Context Agent 78
6.2.2 Security and Privacy 78
6.2.3 Buddy-list Setting 79

References **80**

Appendix Ⅰ: System Installation Guide **83**

Appendix Ⅱ: Structure of Knowledge **85**

List of Figures

2-1 Location Hierarchy 6
2-2 User Mobility in Two Domains of Location 8
2-3 Relationship Model for Deciding Importance of a Message 12
2-4 Context Models 19
3-1 Black-box View of the System 22
3-2 Block Diagram of the System 24
3-3 Server Module Class Diagram 27
3-4 Client Module Class Diagram 30
3-5 System Architecture and Module Interactions 31
3-6 System Topology 32
3-7 User Case Diagram 36
3-8 State Chart Diagram for User Login and Registration 37
3-9 The Sequence Diagram for Call Processing 39
4-1 The Server Welcoming Window 41
4-2 Logging into the Server 42
4-3 Server Main Window 43
4-4 Context Setting Window 44
4-5 Computer Name and Extension Number Setting Window 45
4-6 Relationship Assigning Agent Window 46
4-7 User Rule Assigning Agent Window 47
4-8 User-rule Conflict Resolving Agent Window 48
4-9 Call Simulation Window 49
4-10 Client Welcoming Window 50
4-11 Logging into the Client 51
4-12 Registration Window 52
4-13 Client Main Window 53
4-14 Relationship Setting Window 54
4-15 Buddy-list Setting Window 55
4-16 Schedule Setting Window 56

4-17 User-rule Setting Window 57
4-18 Creating a New Rule – 1. Select Context Condition 58
4-19 Creating a New Rule – 2. Select Caller Condition 59
4-20 Creating a New Rule – 3. Select An Action 60
4-21 Creating a New Rule – 4. Confirmation 61
4-22 Creating a New Rule – Entire Process Diagram 62
4-23 Context Simulation Window 63
4-24 Call Delivery Agent Window 64
4-25 Incoming Call Notification Window 65
4-26 System Management Agent Window 66
5-1 The Design of a Active Badge 68
5-2 A Badge Sensor and Telemetry Network 69
5-3 Primary Interactions Between the Clients and a Server 71

Chapter 1

Introduction

What we call "intelligence" in Artificial Intelligence or "good design" in Human-Computer Interaction amounts to being sensitive to the context in which the artifacts are used. Doing "the right thing" entails that it is "right" under the given context. Many of the frustrations of today's software such as cryptic error messages, tedious procedures to be followed in its usage, and brittle behavior are often due to the software taking actions that may be right for the software's assumptions, but may be unsuited for the user's actual context. One way out of this is to characterize the context and have the software know more about, and be more sensitive to the context. [Lieberman, 01]

Context is a powerful concept in communication. Communication between humans is where is the context implicit. Context can be used to interpret explicit acts, making communication much more efficient. However, there is currently no generally accepted definition of context. This thesis attempts to characterize the context in a domain of communication systems and then apply it to existing communication products. Our goal is to develop a system that acts appropriately when a message[1] arrives, according to the user's current context. The system is highly personalized with flexible options for managing the incoming messages.

A new approach for designing a "context-aware system" is another major issue of our research. Such a system should have a flexible architecture that easily adapts to new ideas or technologies related to contexts. The proposed system in the thesis focuses on voice

[1] Although terms, a 'call' and a 'message', are used interchangeably, a call refers to only a voice message, whereas a message refers to any possible message such as e-mail and instant message.

messages delivered through a telephone system. It may be extended to deal with other message types such as e-mail or instant messages in the future. In fact, the synchronous nature of a phone call limits the choices for possible delivery actions, and that fact makes it easier to apply to other message types with more choices of delivering actions.

Our prototype system, which is called Active Call Delivery (ACD) system, exploits pre-defined sets of context information. There are three categories of context that are used in the system: the *"location"*, the *"activity"* and the *"time"*. The relationship information between a caller and a receiver is used as a part of the knowledge. Having necessary knowledge, we introduce a user preference rule, which gives a user control over the incoming calls. The prototype system extends the currently available PBX[2]-type phone system developed by MITEL Networks.

1.1 Research Focus

System Architecture and Design: A flexible system design for multiple software agents, new concepts of context handling, and distinct methodologies for call handling are our first level research focus. To give such flexibility, the software agents need to have a common medium to share their knowledge. A tuple-based blackboard model provides an excellent opportunity to share knowledge among agents. How the agents communicate effectively through the blackboard system is the key issue to enhance the system's overall performance. The performance, which is measured in response time, is particularly critical for synchronous type of messages such as phone calls.

[2] A PBX (Private branch exchange) is a telephone system within a company that switches calls between users on local lines while allowing all users to share a certain number of external phone lines. The main purpose of a PBX is to save the cost of requiring a line for each user to the telephone company's central office.

Context Classification: There are three major stages in the use of contexts, *information gathering*, *interpretation* and *response generation*. We concentrate on the interpretation stage and assume that reliable context detecting technologies are available. This aspect of context detecting will be simulated in our prototype system.

1.2 Thesis Overview

In the next chapter, the research issues, which are related to the design and implementation of the system, are described in more detail. The major issues such as the system architecture and the context classification are clearly delineated, and other issues such as creating user preference rules and resolving conflicts in a rule-set are explored. Furthermore, associated technologies, required for implementation, are introduced in this chapter.

In Chapter 3, the overall system architecture for a context based message delivery is explained. It also discusses how each module of the system contributes to accomplishing the desired output. Module interactions and knowledge management are covered in this chapter.

Chapter 4 describes the installation and configuration for both a client and a server system. Instructions for using the system, methods for managing knowledge and for creating the rules are explained in detail.

Chapter 5 explains research efforts and existing commercial systems that are related to our research.

Finally, Chapter 6 shows how this thesis contributes to making effective communication and concluding remarks. Some possible directions for future research and implementation are indicated.

Chapter 2

Issues Related to "Context"

2.1 Overview

This chapter lays the groundwork for the thesis by examining aspects of background knowledge. It starts with an attempt to define the "context" for message delivery purposes and explores other key aspects such as the relationships between a caller and a receiver, user preference rules and rule conflicts. Use of "contexts" has a great potential to enhance usability in computing systems. It is particularly important where a system needs the user's attention and reaction. Without knowing the user's current context, which varies dynamically, adapting a system's behavior cannot be fully satisfactory. Our goal is to minimize the interruptions to a user in a message delivery system by making use of the current context of the user.

2.2 Characterization of Context

2.2.1 Context Definition

Several researchers have defined the term "context" in general. Schilit and Theimer [Schilit, 93] defined context as a location, the identities of nearby people and objects. They mostly consider 'where you are', 'who you are with', and 'what equipment is nearby'. Albrecht Schmidt *et al* [Schmidt, 99] defined context as interrelated conditions in which something exists or occurs. They pointed out that the major factors to be considered while processing contexts are the user's social environment and the task being performed; and *physical environment* such as location, infrastructure and conditions. Furthermore, Peter Brown *et al* [Brown, 98] defined context as a location,

the identities of people around the user, the time of day as well as time of year, season, temperature, radiation level and synthesized these data from lower level sensors. In the definitions of contexts by various researchers, we note the following aspects varying from general to specific:

- The parts of a written or spoken statement that precede and follow a specific word or passage, which known as linguistic context.
- A set of facts that surround a particular event, situation, etc.
- The general situation that helps to decide the action to be taken upon message reception.

We view the definition of context according to other researchers as *an abstraction of the collection of relevant background features.*

Among the definitions of others, a definition from Anind K. Dey [Dey, 99] is the most suitable for our purpose. He defined context as: "It is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." Based on this understanding, we note that contexts for message delivery can be categorized as the user's *location, activity, and the time of an event.* For example the following context changes may trigger an action for message delivery.

- When a *location* is changed. (e.g. Enter the office)
- When an *activity* (e.g. meeting, phoning, etc.) is over or get started
- When a *time* has come

2.2.2 Location

We hypothesize that the user's location is an important factor in designing an intelligent message delivery system. In fact, one significant challenge of message delivery systems is using constantly changing a user's locations. When people moves from one location to another, their surrounding environments also change. Changing the location may also lead to changes of available communication resources, connectivity, and cost in the message delivery aspect. Therefore, these changes should be considered as critical factors in deciding an action of message delivery. Detecting a user's location has been a relatively active area of research and a variety of technologies are available for tracking the locations of people. Active Badge System [Want, 92], PARCTabs [Adams, 93], ultrasonic techniques [Ward, 97] and the Cambridge Positioning Systems[3] are the well-known examples of the research related to recognition and use of the location information.

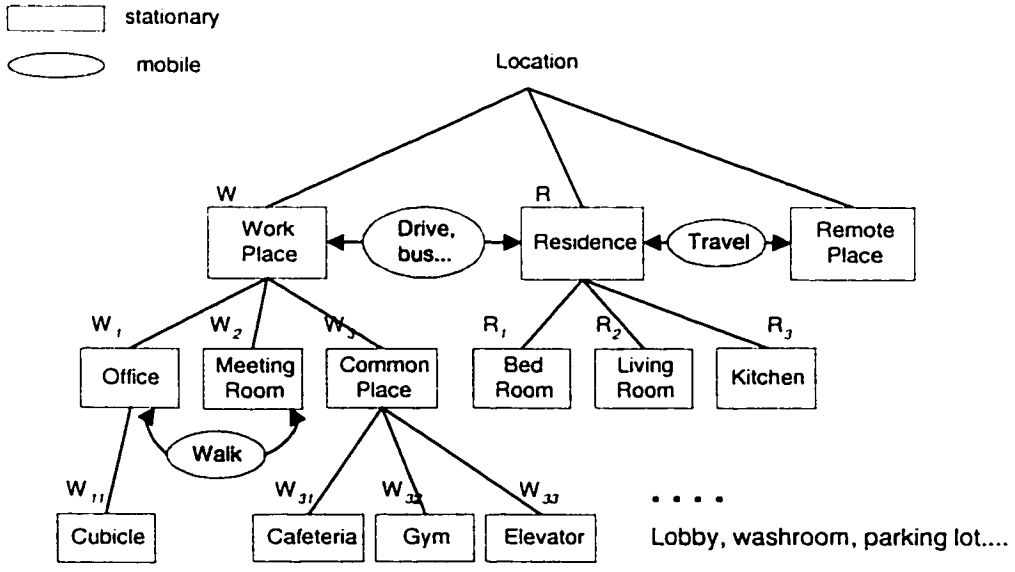


Figure 2-1: Location Hierarchy

[3] Cambridge Positioning Systems Ltd (CPS), http://www.cursor-system.co.uk/

Location information can be hierarchically viewed and expressed by a tree structure reflecting the child/parent relationships among all locations in a specific user domain. A child can have only a single parent, but a parent may have many children. For example, in an office worker's domain, an office worker can be in several sub-locations of his work place such as office, meeting room, cafeteria, etc. Figure 2-1 shows an example of location hierarchy.

Each location has its own properties, which might be useful for the message delivery. The property of a location is information about its purpose, available devices, physical location (e.g. floor *x* section *y*), abstract name of the location (e.g. meeting room), personalized location (e.g. my office), and possible/permissible activities in that location. From these properties of a location, we can choose the best way for delivering a message to that location. Furthermore, the property of a particular location might assist a user to create preference rules to deal with incoming messages. One possible location in an office is a *cubicle* and its property information as an example is as follows:

W_{11} - Cubicle

- Equipped with a NT computer with local IP address, a Unix terminal with terminal name and number, a phone with voice mailbox and display
- Located in floor *x* and section *y*, which is physical and unique location information
- Possible activity: Working or on the phone

Assuming that a user is staying in his cubicle. A NT computer, a Unix terminal and a phone with voice mailbox and display are equipped in the cubicle, and some of them are available (e.g. not busy, logged in, etc) to respond to the message delivery system. When a message arrives, the system checks the availability of equipped devices based on the

location's property information. If the system recognizes the location of a user and the nearby available communication devices, the most appropriate action for an incoming message, which is the least disturbing, might be possible. On the other hand, when the system needs to deliver an urgent message, it can make use of one of the nearby available or interruptible devices that may get the most attention. In addition, possible activities in a certain location, which will be discussed in the next section, will assist in handling incoming messages.

As the location changes dynamically, the mobility of a user becomes important. Basically, mobility is a transition from one location to another, and we need to consider mobility as a separate state when the time taken to move from one location to another is significant or non-negligible. That is, important messaging events can occur in that time interval.

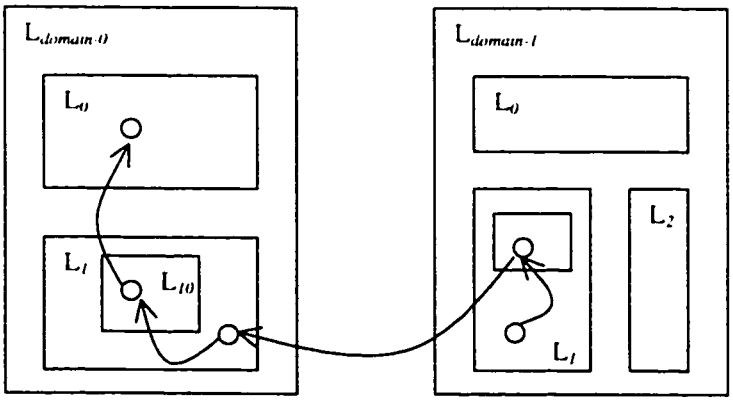


Figure 2-2: User mobility in two domains of location

In the figure 2-2, L_i is a possible location within a domain, and L_{ij} is a sub-location within L_i. The depth of nesting can be extended, as needed. When a user moves from one place to another, it takes time. If the time for moving, T_{move}, is small enough to be

ignored, it is unnecessary to consider it in the message delivery. In fact, if we classify and use all the possible locations in a location domain, mobility can be viewed as a sequence of location changes. However, classifying all the possible locations is probably unnecessary and might even be impossible in some cases. If we classify the domain of office buildings, we may identify and classify major locations in the building such as office room, meeting room, cafeteria, and leave other common places such as elevators, corridors and lobbies unclassified. All the unclassified locations can be viewed as monolithic and processed the same way for message delivery.

When a user moves between domains (e.g. from "work place domain" to "residence domain"), the time taken to move T_{move} might be significant. If time to move from source domain (D_i) to target domain (D_j) is Δ, and if it is much bigger than the average time between two consecutive message arrivals ($\tau_{average}$), the state of "on-the-move" becomes important in message delivery.

$$\Delta \{D_i \rightarrow D_j\} >> \tau_{average}$$

This mobility of a user between domains challenges us to consider the dynamic characteristics of the domains. Suppose, a user moves from D_1 to D_2. If D_1 does not have a firewall and D_2 has, the mobility of user from D_1 to D_2 leads to getting into a firewall that needs to have special operations. In addition, if D_1 has multi-modal communication devices, and D_2 has uni-modality, then the mobility of the user changes the possible modality for message delivery.

2.2.3 Activity

If a system is able to identify what the user is currently doing, appropriate message delivery action with least interruption to the current activity could be possible. A user may engage in one or more activities either concurrently or sequentially within a domain. A definition of an activity should be customized according to a domain since the meaning of an activity may differ from one domain to another.[4] For example, "fishing" may be a "leisure activity" for an office worker, but it should be defined as a "work" for a fisherman.

Interesting questions arise when we describe the activities of a user. What if a user is involved in multiple activities at the same time? We often attend to a phone call while surfing the Internet. Is it really multi-tasking? As Miyata and Norman [Miyata, 86] classified, there are different kinds of activities such as current, foreground, background and suspended activities that exist when people take part in multiple activities concurrently. However, there will always be one and only one activity as the foreground activity. When people are multitasking, only one task can be consciously controlled (Davies et al. 1989).

In fact, people's multitasking is very similar to the software system's multitasking or multithreading with a single CPU. In the case of a computer's activities, the processing time is much faster when compared to the granularity of reaction time of the user, and people do not have any difficulty in perceiving a set of multiple interleaved tasks as if they were concurrent. However, the idea is hard to transfer to people's activity since it is much more complex. It is necessary to approach this issue from a different aspect if it is to be useful.

[4] Definition of activity is often interrelated to the definition of task in work domains.

We classify the people's activities in a domain and then analyze them to find what activities can be performed concurrently. For example, people can perform both "surfing on the Internet" and "talking on the phone", but it's not possible to do both "driving a car" and "walking in the hallway". When two activities, A_1 and A_2, can be performed concurrently, we denote them symbolically as:

$A_1 \parallel A_2$

When two activities, A and B, should be done in sequence such as attending two phone calls in a "call waiting case", the sequential nature is denoted by a semicolon as:

A_1; B_1; A_2; B_2; A_3; B_3 (where A_i and B_i are partial tasks in whole activities A and B)

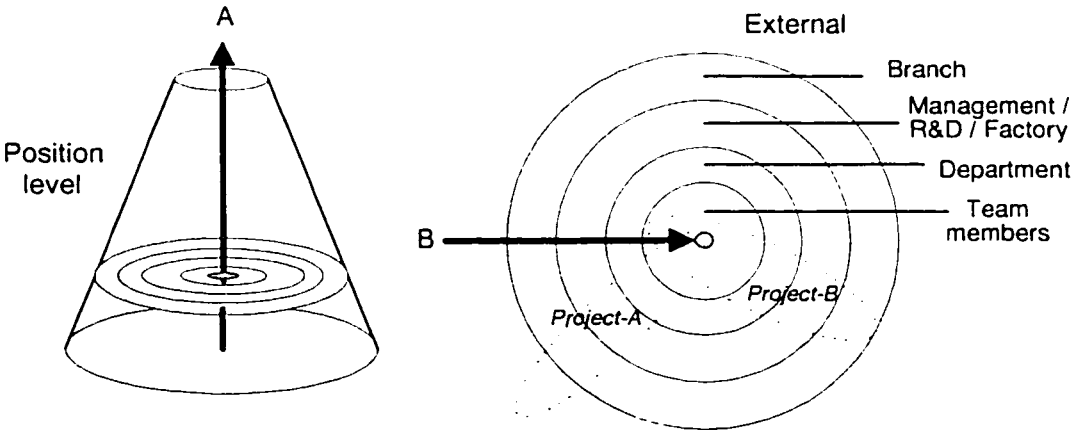
2.2.4 Time

Time is another important aspect of context, especially for message delivery. Timely delivery of a message would be more desirable than blindly delivering messages as they arrive originating from different sources. Time of delivery can be set by the sender, or the receiver, or both. One possible functionality is that a system allows a sender to set a time to deliver a message. For example, Jerry wants to send a voice message (e.g. happy birthday message) for Tom to be delivered at a specific time in the future. Jerry records a voice message and sets the time for delivery. When the time comes, Tom's phone will ring and play the message. If it can be combined with the receiver's location information, the effectiveness of the message delivery might be improved. For instance, the Jerry's 'Happy Birthday' voice message can be sent as soon as Tom arrives in his office in the morning.

A receiver's time schedule can be another way of using time context. If a user's lunchtime is scheduled from 12 a.m. to 1 p.m., the system can assume that the user is having lunch during the time period. This information can be used to define user preference and to customize delivery actions.

2.3 Relationship between message sender and receiver

Delivering a message involves a sender and a receiver. In a sociological context, a relationship can be viewed as a social connection between people. This view excludes any emotional factors, which are beyond the scope of this thesis. One of the most important reasons for building the relationship model is that it is one of the considerations in deciding the importance of delivering of an incoming message. From the receiver's perspective, the relationship with the message sender can determine whether it is an important message.



(a) A view of a user from authority level (b) A view of a user from peer

Figure 2-3: Relationship Model for Deciding Importance of a Message

In the figure 2-3 (a), the gray dot in the middle of circle represents a user and the cubic volume of body represents a number of people in his company, assuming that people or

positions in a higher level of authority are fewer. The arrow 'A' indicates the possibility of increasing level of the importance of a message according to the level of authority. The figure 2-3 (b) represents people or positions in the same level of authority in the user's company while the gray dot represents the user as a center of relationships. The arrow 'B' in the figure 2-3 (b) indicates that a message from the person, who has a closer working relationship with a user, is assumed to be relatively more important within a cross section of the "core". The importance of a message from the people who work for the same project(s) may be weighted.

Family members, friends and other people in this domain have different weights according to their own domain characteristics. Other factors such as the content of the message and the behavioral history of the user towards the message source or the message content also contribute to decide the importance of a message.

2.4 User Preference Rule

The user preference rule is one way to control the delivery of incoming messages. The contexts and the relationship information feed the conditions of the user preference rules. The rules consist of conditions and a following action.

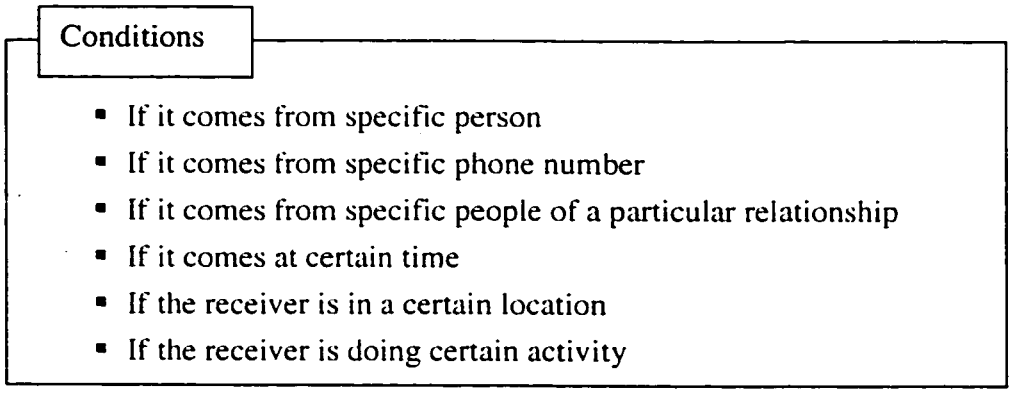
<rule name>

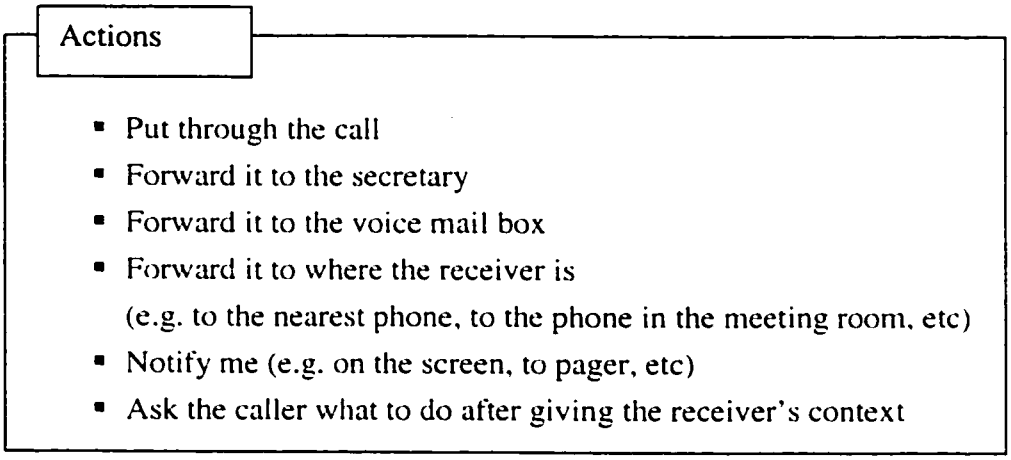
"IF" <assertion> & ... & <assertion> "THEN" <action>

<degree of certainty>

The *<rule name>* represents a sequence number or unique name of the rule to distinguish it from other rules. The *<assertion>* is one of the 'facts' as a condition of the rule such as "The user has PDA," or "The user is in the meeting room." The *<action>* is the decision that should take place if all the assertions are satisfied. The *<degree of certainty>* may be used for finding the most appropriate rule. The possible considerations for setting the

conditions and actions of the user preference rules for phone call delivery are the following:





2.5 User Preference Rule Conflict

The user preference rules are considered to be in conflict when two or more user rules match in a certain circumstances and the actions of the rules are incompatible or opposite to one another. Suppose, a user has the following rules:

User Rule 1: If a call comes from 'Family'

Then 'Ask the caller what to do'

User Rule 2: If a call comes from 'John' (suppose John is grouped as Family)

Then 'Forward it to the voice mail box'

These two rules are in conflict when John calls the user since both conditions are satisfied and the following two actions are incompatible. As choices of conditions and following actions expand in a system, it becomes a significant issue to detect and resolve the rule conflicts.

2.5.1 Detecting Conflicts

- Configuration-time Detection (Avoidance)

  The system can prevent the possible conflicts in the first place. A well-designed GUI can be used to solicit user input in minimizing the possibility of rule conflicts. Furthermore, the user interface program can check the conflicts with existing rules when the user creates a new rule. It reduces conflicts, but the dynamics of the user contexts, relationships and rules do not prevent all possible cases of conflicts. Because of this, we also need to have a run-time detection method.

- Run-time Detection

  To detect rule conflicts during run-time, the rules must be examined with the current context of a user when a messaging event occurs. The conditions of the rules are then being matched with the user's dynamic contexts. Since only one action should be possible, the rule and its following action to be taken should be unique. If there is more than one rule that may be triggered, a selection process should take place. The methods for the selection process, "How to select one rule to trigger in the current context?" constitute a major issue for resolving such conflicts.

  If triggering actions are intermediary, one rule firing may cause another rule to fire. This type of conflict is called a "trigger chaining conflict" [Graham, 01]. Suppose we know not only which conditions and actions a rule contains, but also which rules a

particular rule may trigger. For example if the execution of R_1 results in an event that is in the condition of R_2, then we can say that R_1 potentially triggers R_2. There is another possibility for a conflict that if a rule R_i triggers a rule R_j which triggers a rule R_l etc., until finally a rule R_t is triggered which triggers R_i again. This will causes a cyclic behavior. This means that rules are constantly being triggered, and may never stop executing. Not only this but depending on the rules being triggered, if a job is being passed onto the next rule, then the job will never be completed because no rule ever tackles the problem. For example say R_1 owned by Bill states that upon receiving a work-related e-mail, the e-mail should be forwarded to John, and R_2 owned by John states that upon receiving a work-related e-mail, the e-mail should be forwarded to Bill. In this case, if either Bill or John ever receives a work-related e-mail one rule will trigger the other, which then will trigger again the first rule, and this cycle continues. The e-mail never gets read, and many CPU cycles are wasted. Therefore, it is essential that this behavior is detected in the rules and reported.

2.5.2 Resolving Conflicts

Chomicki, Lobo and Naqvi [Chomicki, 00] proposed a way to resolve conflicts by assigning priorities for each rule from the user for the user-set-rules. A more appropriate approach for this system is what Lupu and Sloman [Lupu, 97] have proposed. The first method is to always give priority to negative rules. Suppose, if one rule gives positive authorization for an action and another rule gives negative authorization for the same action, the negative authorization overrides the positive one. The second method is to give each rule an explicit priority. This can then be used to give rules precedence in the event of a conflict. The third method is to examine the distance in the class hierarchy of the object being managed from the rule. Intuitively, the more specific rule gets a higher priority.

2.6 Associated Issues

2.6.1 Message Delivery Classification

Messages can be classified by their types of intended delivery. Three possible categories of message delivery will be discussed; *push versus pull delivery types*, *store-and-deliver versus deliver upon message arrival*, *and full message versus message abstract.*

- Push versus pull delivery types

  A message delivery type can be classified according to whether a receiver of a message is active or passive object. When a receiver is passive, the type of message delivery is "push type". A message is conceptually "pushed" to a receiver by a message sender or by a messaging system. The examples of this type are e-mail, phone, and instant message. On the other hand, a receiver actively seeks out an interested message in the pull type message delivery. A message is "pulled" by a receiver. In this pull type of the message delivery, there are two distinctive kinds in terms of a type of available message. It can be a continuous and dynamic stream of messages as we have seen from broadcast systems. Otherwise, it is a static and context-dependent message, which we are interested in. Some applications make use of this pull type and location dependent message delivery. Examples are Stick-e note [Pascoe, 97], MemoClip [Beigl, 00], and CyberGuide [Long, 95].

- Store-and-deliver versus deliver upon message arrived

  Whether a message is saved in the repository or not is also an important characteristic of message delivery. All the asynchronous messages are saved in a repository and wait for later perusal. Semi-synchronous[5] and synchronous messages are not saved

[5] In the semi-synchronous mode of message communication, the response time d_t is *predictable*, but *not* strictly bounded. The acceptable time delay, τ, is usually greater than that in synchronous communication. $T_s << \tau << \tau_{max}$ (e.g. $\tau_{max} = 10$ seconds)

but are delivered immediately. Note that if we use an intelligent agent, messages in the repository no longer wait until it is checked, but they can be delivered actively.

- Full message versus message abstract

  In some cases, receiving entire messages creates overhead for a receiver. Moreover, messages in audio, video, and graphic forms may overwhelm a user. A summary of those types of messages might beneficial for a user to get instant knowledge without observing the whole contents. In addition, abstracted messages may be useful for searching. When a called number is displayed on a LCD screen, the receiver knows "who is calling" and possibly presumes "what the call is about" even before answering the call. If more information is available such as the purpose and subject of the call, the receiver or his agent can have more idea about the entire message. Therefore, if we can collect and provide the user with what the incoming message is about, the receiver can have better control. Especially for a message that was saved in a repository, it is easier to collect necessary information. We may see the message such as "There is a phone call from George regarding your demo program", or "There is an instant message from Michael to confirm your reservation for tonight's dinner party". The receiver of such detailed messages might have a better idea of what to do about the message in the light of his current activity. Note that such summarized messages can also be useful when a user browses through a log of messages that arrived, while the user was not available.

2.6.2 Context Model and Representation

Context modeling uses different parameters according to the areas of research. In the cognitive science, contexts can be a number of stimuli – light, temperature, movement, sound, etc. – these are detected by sensors. Some models are proposed to process context in the computer system. In most cases, context is both spatial and temporal since

stimuli are physical and must be attended in sequence. In the limiting case, a context can consist of a single event such as the presentation of a stimulus and the time of occurrence. In this case, the context essentially acts as a stimulus trace.

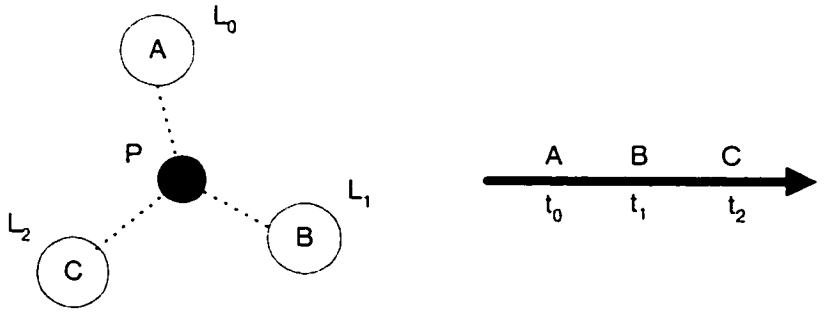


Figure 2-4: Context Models

Figure 2-4 illustrates two types of contexts. The left one represents a spatial context given by three stimuli at three different locations. The other represents a temporal context consisting of three events A, B and C, occurring sequentially.

There are two approaches to represent contexts, which may be beneficial for our research. A formal approach helps to build a structural model of contexts. McCarthy and Sasa Buvac [McCarthy, 95] have tried to formalize contexts in general. Their goal of formalizing contexts is to allow simple axioms for common sense phenomena, e.g. axioms for static blocks-world situations. Their main logical formulas are sentences of the form:

$$c' : ist(c, p)$$

It is to be taken as assertions that the proposition p is true in the context c, itself asserted in an outer context c'. To this end, formulas $ist(c,p)$ are always considered as themselves asserted within the context, i.e. we have something like $ist(c', ist(c,p))$. Suppose we have the following question. "Who is Holmes?" If the question is in the context of

Sherlock Holmes stories, the answer is "Holmes is a detective." If the question is in the context of U.S. legal history, the answer is "Holmes is a Supreme Court Justice." and so on. The formulas for these expressions are:

c0: ist(context-of("Sherlock Holmes stories"), "Holmes is a detective"*)*

c0: ist(context-of("U.S. legal history", "Holmes is a Supreme Court Justice"*)*

The other approach to represent contexts is defining the context of the eXtensive Markup Language (XML). This way, a hierarchy of contexts can be expressed with tags.

<context>

<location>

<office>

<meeting room>Conference Room</meeting room>

</office>

...

The tags and contents for representing context can be expressed abstractly. To interpret the abstractions, semantic interpretations or associations are needed. Speculating how the expressions of matching and tags can be extended might be useful. The system can trigger the action if all, more than one, or none is matched in the set of conditions. The tags represent both the context itself such as <location> and <activity>, and the restriction of the context such as <at>, <with>, <during> and so on. More specifically, the matching algorithm should prevent re-triggering while the user is in the context by either checking before triggering or a de-triggering when the user leaves the context.

2.7 Summary

This chapter has discussed contexts for message delivery purposes. The definition of context reveals a potential to enhance usability in a messaging system when it is included as a part of decision-making processing: where, when, and how to deliver. We explore a "relationship" between a message sender and a receiver to decide an importance of incoming message. An approach for controlling incoming messages in this thesis is a user preference rule based on contexts and relationship information. Possible conditions and actions of a user preference rule are introduced, and detecting and resolving rule conflicts are investigated. Facts covered in this chapter are used in our prototype system described in the next chapter.

Chapter 3

A System for Context Based Message Delivery

3.1 System Overview

The Active Call Delivery (ACD) system is a context-aware and rule-based system that helps a user to receive calls at the right time and in the right manner. Our implementation goal is restricted to design an intelligent means of a "voice message delivery system" that allows effective communication according to a receiver's preferences and a receiver's current contexts.

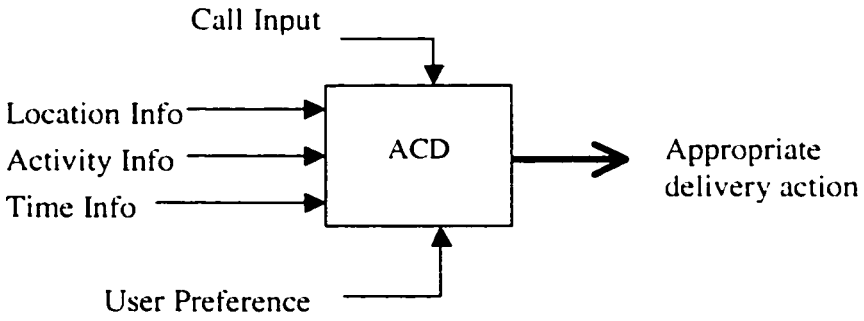


Figure 3-1: Black-box View of the System

The figure 3-1 shows the system's inputs and outputs while the system is treated as a black box. When a call arrives (Call Input), we will take advantage of both a user's current context characterized by 'Location Info', 'Activity Info', 'Time Info' and a user's preference in that context to decide the 'Appropriate delivery action'. An algorithm for choosing the appropriate action forms the core part of the system. Under appropriate cases, we involve the end-user in deciding the appropriate action. Making a decision with limited or incomplete context information leads a user to distrust the system since the system's decision may be right given the system's assumptions, but wrong as the user has

other expectations. Therefore, the system should allow a user to create his/her own preference rules, which includes a delivery action, and the system should not decide the action itself.

The knowledge such as user's context information, profile, and user preference rules are saved in the TSpaces[6]. TSpaces is also used as a network middleware for communicating among software modules of the system. A succinct description of TSpaces would be to view it as a network communication buffer with database capabilities. It enables communication between applications and devices in a network of heterogeneous computers and operating systems. TSpaces provides group communication services, database services, URL-based file transfer services, and event notification services. It is implemented in the Java programming language and thus it automatically possesses network mobility through platform independence, as well as a standard type of representation for all data types. The TSpaces system is appropriate for any application that has distribution or data sharing requirements. It can perform many of the duties of a relational database system without imposing an overly restrictive (and primitive) type system, such as a clumsy user interface or a severe runtime memory requirement. In a sense, it is a database system for the common requirements. In our prototype system, the TSpaces Server is the medium between the Application Level and the knowledge storage. It is responsible for managing all tuple-based knowledge transactions, which include read, write, update, take and scan. It is also responsible for event handling, such as registration and notification of events.

The MiTAI Gateway, developed by MITEL Networks, facilitates communication to the MITEL telephony servers for processes that are not based on the "C" development language. Full MiTAI functionality is not supported but limited capabilities like making

[6] http://www.almaden.ibm.com/cs/TSpaces/

calls and transferring incoming calls are supported. The MiTAI Gateway is a Windows based system that can be executed on any Windows platform. It can manage a single socket connection from any other process on a network and it supports a limited session protocol. It was originally developed to support the Personal Policy Agent (PPA) demonstration that is based on the IETF Call Processing Language (CPL). [CPL, 00] That process was developed using the Java language and it can easily open a socket connection to the gateway rather than creating JNI interfaces to MiTAI directly. The MiTAI Gateway Server is the intermediate system between the PBX and the ACD's Application Level subsystem. The Application Level subsystem registers an event to the MiTAI Gateway server for the purpose of monitoring incoming calls.

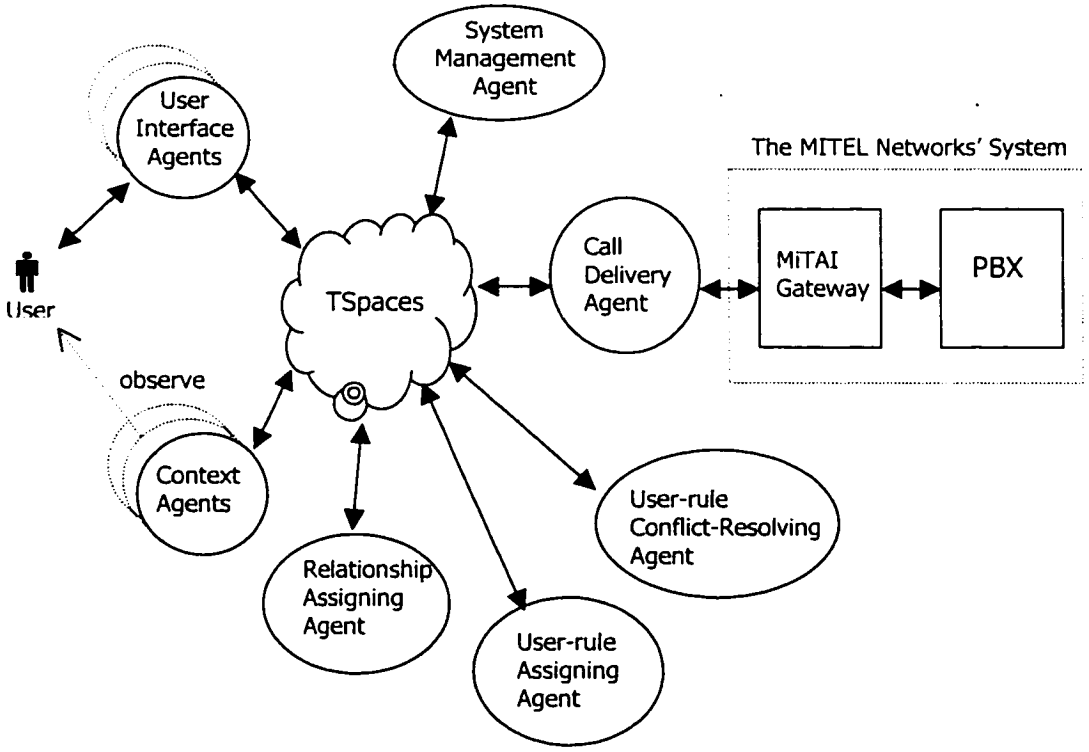


Figure 3-2: Block Diagram of the System

The ACD system is based on a client-server architecture. An ACD client communicates with a TSpaces server and a MiTAI Gateway server. An ACD server is software that runs on any machine that has access to a TSpaces server. The server system consists of user interfaces and a collection of "system agents". The user interfaces provide an introduction window, a login window for a system administrator, a context window for managing a hierarchy of contexts and a call simulation window for simulating a phone call. Each system agent contributes to call processing and it has own responsibilities: The Relationship Assigning (RA) agent[7] is responsible for acquiring the relationship between a caller and a receiver, and assigning it to a relevant data field for call processing. The User Rule Assigning (URA) agent is responsible for extracting all the matching user rules according to the conditions of each rule and the current context, and assigning them to a relevant data field for call processing. The User-rule Conflict Resolving (UCR) agent is responsible for resolving the conflict that might be present in the assigned rules. These agents do not have to be installed on a particular machine, but can be distributed over a network of machines, which have access to the TSpaces server.

The client system consists of user interfaces and user agents. The user interfaces provide an introduction window, a login window for a registered system user, and a registration window for a new user. Knowledge management is an important part of the user interface on the client system. A user can create or manage personal information such as a buddy list, relationship information, a schedule and a user preference rule. Context simulation is the part of the user interface used for testing the prototype. The client machine has two types of agents: The Call Delivery (CD) agent and the System Management (SM) agent. The CD agent acknowledges events, which are generated by the Call Monitor, in the TSpaces. The Call Monitor is a direct interface with the MiTAI Gateway, and creates an

[7] There is no universally accepted definition for the term "agent". It generally refers to a software system that is autonomous, act asynchronously, stores knowledge explicitly, makes use of it, learns over a period of time, etc. In this thesis, the agent software possesses most of these attributes but not the "learnability".

event to be fed into the TSpaces for starting call processing by the CD agent. Then the SM agent acknowledges the event from the CD agent, and distributes the call processing to agents on the network. Although each agent has distinct services, both the server and the client have certain common modules as per the object-oriented design. The following section explains these common object modules and the other modules.

3.2 Architecture

The architecture section deals with issues of organizing the system into components and assigning functionalities to these components, demonstrating how they communicate and share knowledge.

3.2.1 User Interface

The user interface consists of three windows: message windows, notification windows, and control windows. They include buttons, trees, tables and text fields. The user interface for the client system includes windows for user login, registration, user preference rule setting, context simulation, and a display of messages for assisting a user. The server's user interface includes windows for administrative login, context setting and call simulation, and other pop-up message windows.

3.2.2 Event Handler

The Event Handler subsystem is a monitoring daemon that resides between the user interface and the Application Level subsystem. It waits for physical events to arrive from the user interface, such as mouse clicks, and directs them to the appropriate application module. The development tool, Java, provides embedded event handlers, such as ActionListener, for this purpose.

3.2.3 Application Level

The Application Level is the core of the system and consists of multiple agents that provide services for a client as well as for a server. All the system transactions, functionalities, and knowledge management are performed within this subsystem.

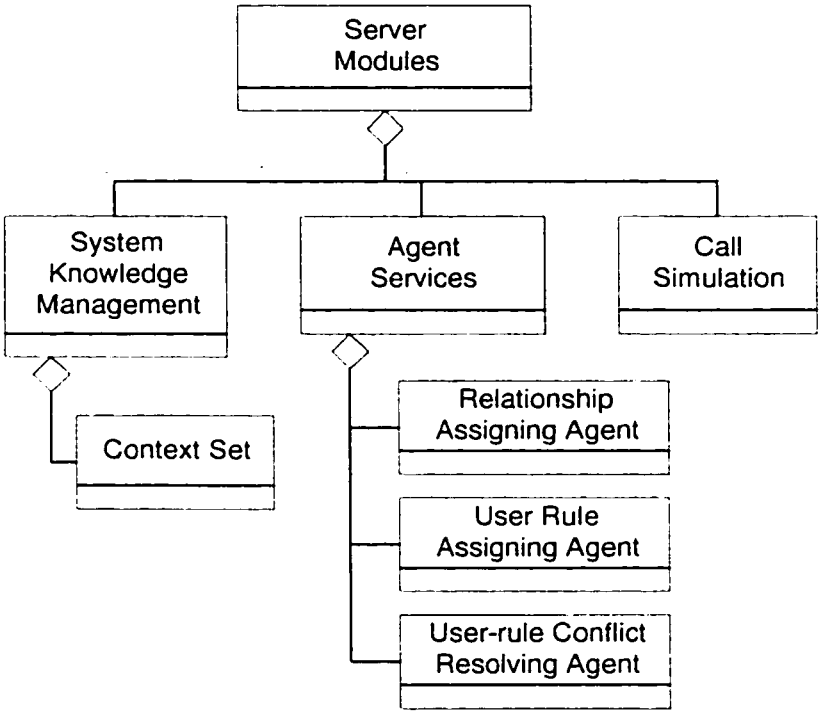


Figure 3-3: Server Module Class Diagram

The server modules are divided into three major parts: System Knowledge Management, Agent Services and Call Simulation. The current version of the system provides a Context Set sub-module for the System Knowledge Management module. It allows an authorized administrator to create or modify a context hierarchy such as location and activity. The Agent Services module consists of three distinct agent modules: The Relationship Assigning (RA) agent, the User Rule Assigning (URA) agent, and the User-rule Conflict Resolving (UCR) agent. Our goal is to have a flexible implementation of these agents. The status of these agents is monitored in order to know

their availability. Network connectivity may affect their availability. The system acquires agent's status by examining a corresponding status tuple in the TSpaces. The status tuple consists of "name", "priority" and "availability" fields. Each agent is responsible for updating its status tuple in the TSpaces. The procedures for updating a status tuple consist of taking the status tuple and rewriting it with new status information in every second. A tuple can be set time to expire. Upon expiration, a TSpaces server removes the tuple from the TSpaces. The expiration time for a status tuple is three seconds, so if an agent fails to update the tuple three times consecutively for any reason, there will be no status tuple for the corresponding agent in the TSpaces. The system assumes that an agent is abnormal if there is no status tuple for the agent, or if the "availability" field in the status tuple is set as "abnormal". The time gap between the one-second it takes to update the status tuple and the three seconds allowed before the status tuple expires may prevent unnecessary status toggling by temporal network disturbances.

Each agent is also responsible for registering an event into the TSpaces to communicate with client machines. Whenever an awaited tuple is written into the TSpaces, a TSpaces server notifies this to an agent that registered for the event. Generating an event and getting notification of the event from the TSpaces forms a two-way communication-and-acknowledgement between agents.

The Relationship Assigning (RA) agent is responsible for responding to a relationship-assigning request from a client's SM agent. The request from a SM agent contains caller and receiver information. The RA agent assigns the relationship between the user and the caller according to the user's buddy-list.

The User Rule Assigning (URA) agent is responsible for responding to a user-rule-assigning request from a client's SM agent. Upon request, the URA agent retrieves both the relationship information and the user's current contexts. The relationship information is a relationship between the caller and the receiver, set by the RA agent. The user's current contexts are the user's location, the current time with the user's schedule, and the user's activity.

- Who is calling?
- Where is the user?
- What the user is doing?
- When is it?

The User-rule Conflict Resolving (UCR) agent is responsible for responding to a client's SM agent for the user-rule conflict-resolving request. The request contains user rule information that is assigned by the URA agent. The UCR agent selects one rule that is the most specific among the assigned rules. If the conditions of a rule are more specific, that rule tends to be more desirable by the user when they are matched. For example, when Bob (who is categorized as a friend) calls, the condition "If Bob calls" is more specific than "If my friends call". When user set a rule for a specific person among his many friends, user may expect a delivery action from "If Bob calls" condition, not from "If my friends call" when both are matched. Furthermore, the more conditions a rule has, the more specific a rule is considered to be. The Call Simulation service is provided for testing without connecting to the MiTAI Gateway.

The client modules are divided into three subsystems: User Knowledge Management, Agent Services, and Context Simulation. A user can manipulate personal knowledge through the User Knowledge Management module.

The Call Delivery (CD) agent is responsible for communication with the phone switch through a MiTAI Gateway. It registers events to a MiTAI Gateway and waits for the notification of an incoming call for a user. When the notification arrives, the CD agent sends a request to the SM agent for further processing and waits for response. This response from the SM agent contains an action to be taken as a result of an entire call processing. Then the CD agent is responsible for requesting the selected action to the MiTAI Gateway.

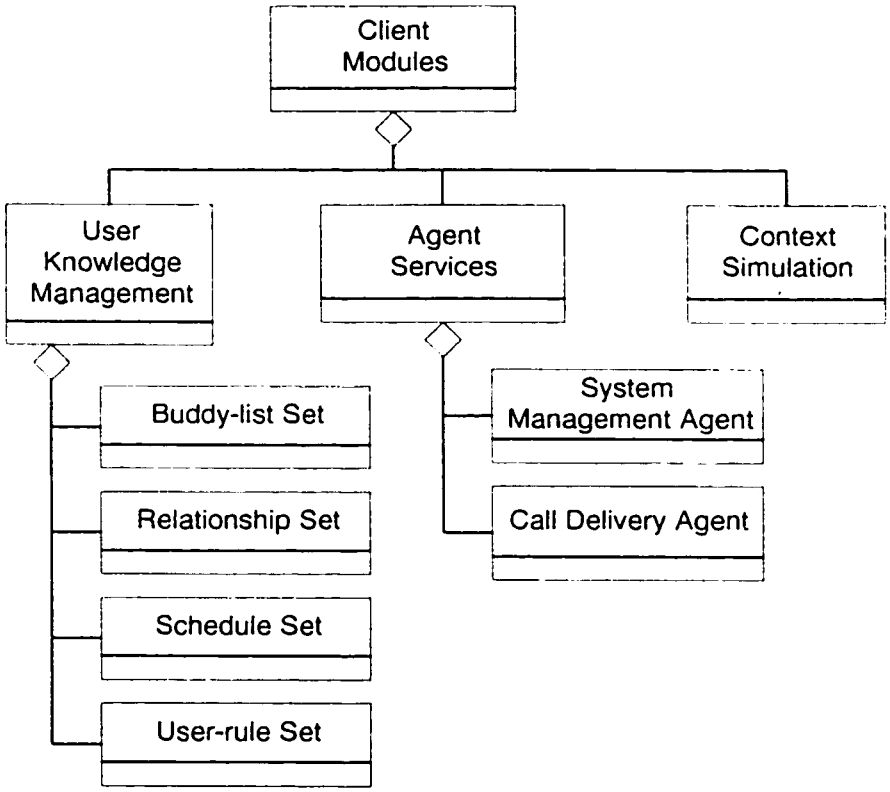


Figure 3-4: Client Module Class Diagram

The System Management (SM) agent is responsible for managing other agents' status and sequencing the call processing actions according to the system agents' priority. When the CD agent requests call processing, the SM agent scans the agents' status tuples in the TSpaces and creates a sequence table according to their priority. It sends a

processing request to the highest-priority-agent and waits for a response. When the SM agent receives a response from the lowest-priority-agent, it sends an information tuple back to the CD agent.

The Context Simulation module serves as a context agent that detects, interprets and updates the user's current contexts dynamically. A Context Simulation window includes all the possible contexts, which are set by a system administrator, and a user selects from them. The entire module interactions are shown in figure 3-5.

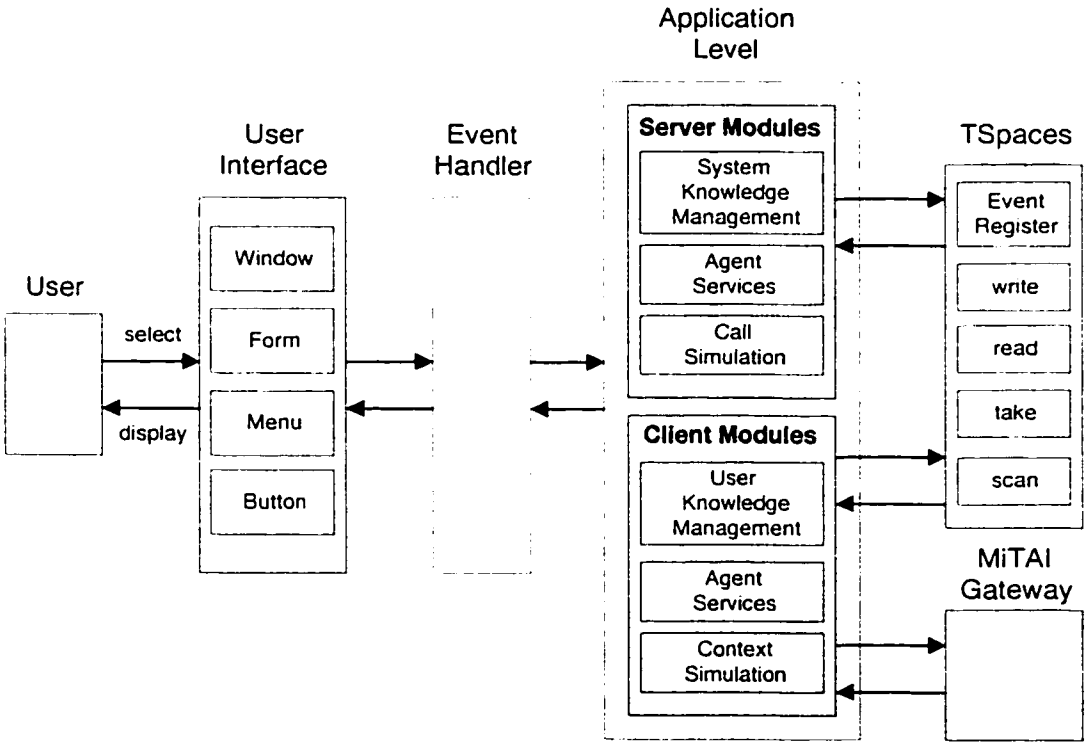


Figure 3-5: System Architecture and Module Interactions

3.2.4 Topology

The following figure shows the layers and partitions of the architecture. The User Interface is built on a Windows Platform, and it interacts with the Application Level

through an Event Handler. The Application Level subsystems use a TSpaces server as a medium to communicate with each other, and access the Knowledge Engine.

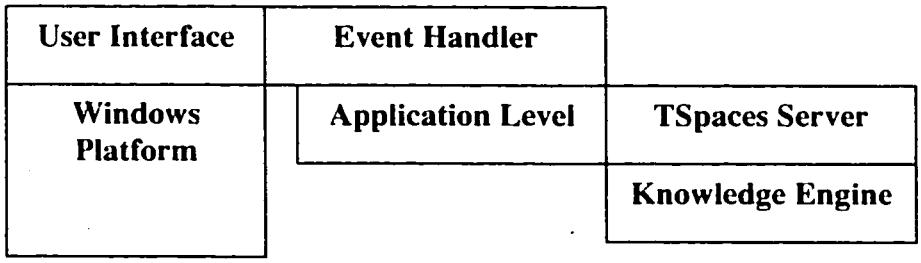


Figure 3-6: System Topology

3.3 Knowledge Categories[8]

All the knowledge, which includes user information, user rules, the user's current context information, and call information, is stored in the TSpaces. The unit of storage is a tuple, which represents the knowledge for our system with each tuple containing one or more fields.

3.3.1 User Information

User Information consists of basic user information, relationship information, a buddy list, user preference rules and the dynamically varying user's current context information. It is stored in a tuple named "UserProfile" and the structure is as follows:

{"UserProfile", id, user-info, relationship, buddy-list, user-rule, context}

- *"UserProfile":* the name of this tuple
- *id:* user identification

  This is a unique identification for a user in this system.

[8] The details of knowledge structure are shown in the *Appendix II: Structure of Knowledge.*

- *user-info:* basic user information

  This field contains basic information about the user: password, name, phone numbers, and the user's time schedule. The phone numbers are extension phone numbers within PBX, such as my office and secretary's phone numbers. (e.g. 4001) This field also contains the schedules of the user. Schedules for lunches and meetings will be input by the user.

- *relationship:* relationship information

  A user defines the relationship hierarchy using the user interface.

- *buddy-list:* a buddy list

  A user can add any person as his/her buddy to his/her buddy list. The buddy list contains information about persons such as name and phone number, as well as their relationship to the user.

- *user-rule:* a user preference rule

  A user creates his/her personal preferences for handling incoming calls by the user interface. The conditions in a rule can make use of the contexts, the buddy list and a relationship selected from the relationship information hierarchy.

- *context:* context information

  The context determining parameters, which will be used in the system, are location, the user's current activity and the present time. The location and activity contexts have a hierarchy, so that they may have sub-contexts. The current context information of a user can be either a real context or a simulated context set by the user. The real context information is updated by the context agent(s) and the simulated context, on the other hand, is set and controlled by the user. The simulated context is designed to override the real contexts if so desired by the user. The hierarchy of the location parameters is defined by a system administrator. Suppose, if a location's properties are coupled with the phone number, the system can deliver the user's call to the phone nearest to the user's current location.

In the context information, "activity" is another aspect that may be useful for the purpose of message delivery. There are two kinds of activities can be defined. Some activities can be automatically detected by the system, whereas others can only be simulated and set by a user. For example, the system is able to know whether the user is 'on the phone', but it is hard to judge if the user is 'busy at work' or 'having a break'. Therefore, detectable activities will be updated by the system automatically, and others should be set by the user. A receiver's time context will be set according to his time schedule. For example, if user's lunchtime is scheduled from 12 p.m. to 1 p.m., the system can assume that the user is having lunch during the time period.

3.3.2 Call Information

The call information is a tuple that an agent shares to communicate with other agents for processing an incoming call. Therefore, it contains all the necessary data fields for caller information and user preference rules. Agents take the "Call" tuple from the TSpaces and update it according to their responsibility. (e.g. the RA agent assigns relationships between the caller and the receiver, the URA agent assigns all the appropriate user rules, and the UCR agent resolves user rule conflict by selecting only one user rule)

{"Call", dest-agent, source-agent, id, call-info, user-rule}

- *"Call":* the name of this tuple
- *dest-agent:* the destination agent that is expected to receive this tuple

The *"Call"* field and the *dest-agent* field are used when agents register an event in the TSpaces server.

```
Tuple template = new Tuple("Call", "SMAgent", new Field(String.class),
                           id, new Field(String.class), new Field(String.class),
                           new Field(String.class));
seqNum = ts.eventRegister(TupleSpace.WRITE, template, this, newThread);
```

The above is a part of the SM agent event registration routine to the TSpaces server. It requests the TSpaces server to notify the SM agent when a tuple with the first field is *"Call"*, the second is *"SMAgent"*, and the fourth is user *id*. '*new Field(String.class)*' means that any value will be acceptable for this field.

- *source-agent:* the source agent that sends this tuple
- *id:* user identification
- *call-info:* This field contains basic information of both the caller and the receiver such as phone number, name and relationship information between them.
- *user-rule:* matched user rule(s) assigned by the agents

3.4 Module Interactions

Module interactions show the behavior of the classes, the modules, and the system as a whole. They describe how the components of the system interact by message passing, function calls, and by sharing the state information. The components interaction is best revealed by means of a sequence diagram and a state chart diagram. These two diagrams of the object-oriented methodology are considered standard representations of system behavior. We use Unified Modeling Language (UML) notation to show these diagrams.

3.4.1 Use Case Diagram

The following diagram sketches "use cases" of the system with the corresponding actors, mainly the application level subsystem. It is designed at a high level of abstraction.

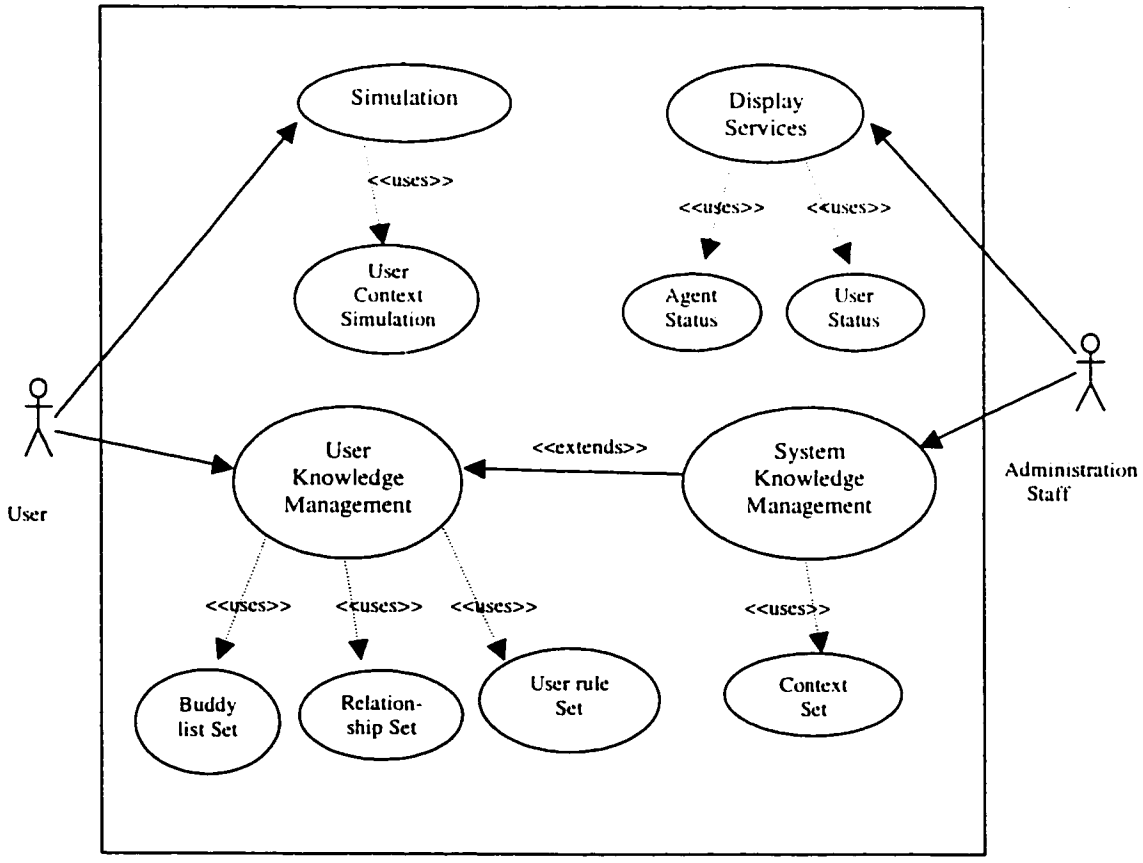


Figure 3-7: Use Case Diagram

3.4.2 State Chart Diagram

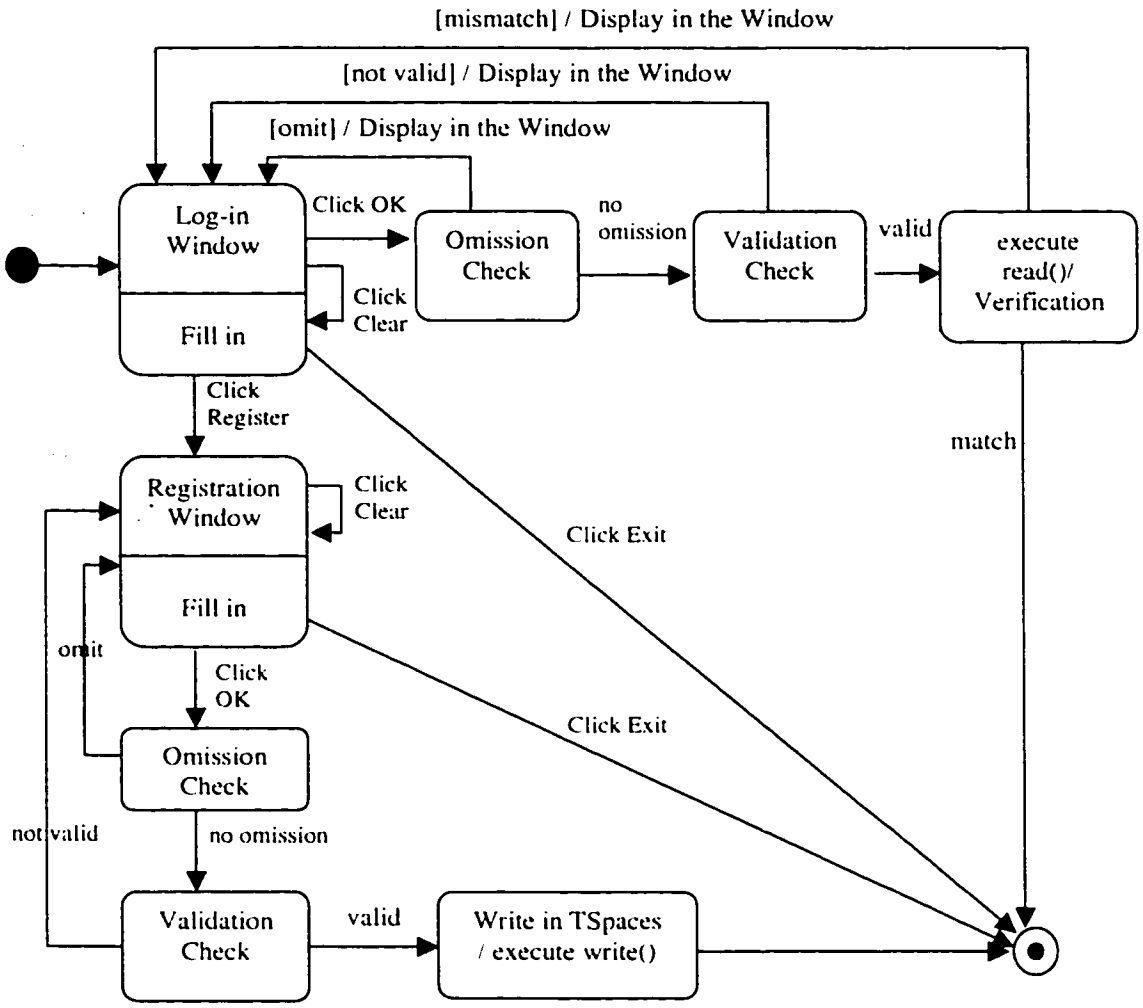


Figure 3-8: State Chart Diagram for User Login and Registration

In order to use the system, including the server system for administrators and the client system for users, a person should be authorized. First time users must register by clicking the "Register" button in the "Log-in Window". Registering users will provide crucial information for using the system such as user-ID, password, name and phone numbers. Every field should be filled in without omission before clicking the "OK" button for submitting. Once it is submitted, the system checks if each field has the right

length and is meaningful. The user-ID should be less than 10 alphabetic characters; the password should be less than 10 numbers and/or letters. The name field should be less than 20 characters and phone number fields allow only numbers. If the "Validation Check" stage in figure 3-8 is successful, the system writes information to the TSpaces by executing the "write()" operation. The user registration processes are finished when the system successfully writes the user's information into the TSpaces.

Registered users and administrators need to be authenticated before using the system. The fields for user ID and password in the "Log-in Window" have to be correctly filled in, and then the "OK" button clicked. If both fields are filled without omission, the system checks and validates of each field. The validated user-ID and password pair should be matched with those in the TSpaces. The system obtains the stored information from the TSpaces by executing the "read()" operation and then compares them for validation. The login processes are finished when either the user clicks the "Exit" button or there is a match between the input user ID and password pair with the pair already stored in the TSpaces.

3.4.3 Call Processing

The call processing sequence is shown in the figure 3-9. It starts when the MiTAI Gateway notifies an incoming call to the CD agent. The CD agent then writes the <call> tuple to the TSpaces server as a call-processing request. The TSpaces server notifies the <call> tuple to the SM agent. The SM agent reads the information about the status of server agents when it receives the notification, and then distributes the processing to the server agents according to their priority.

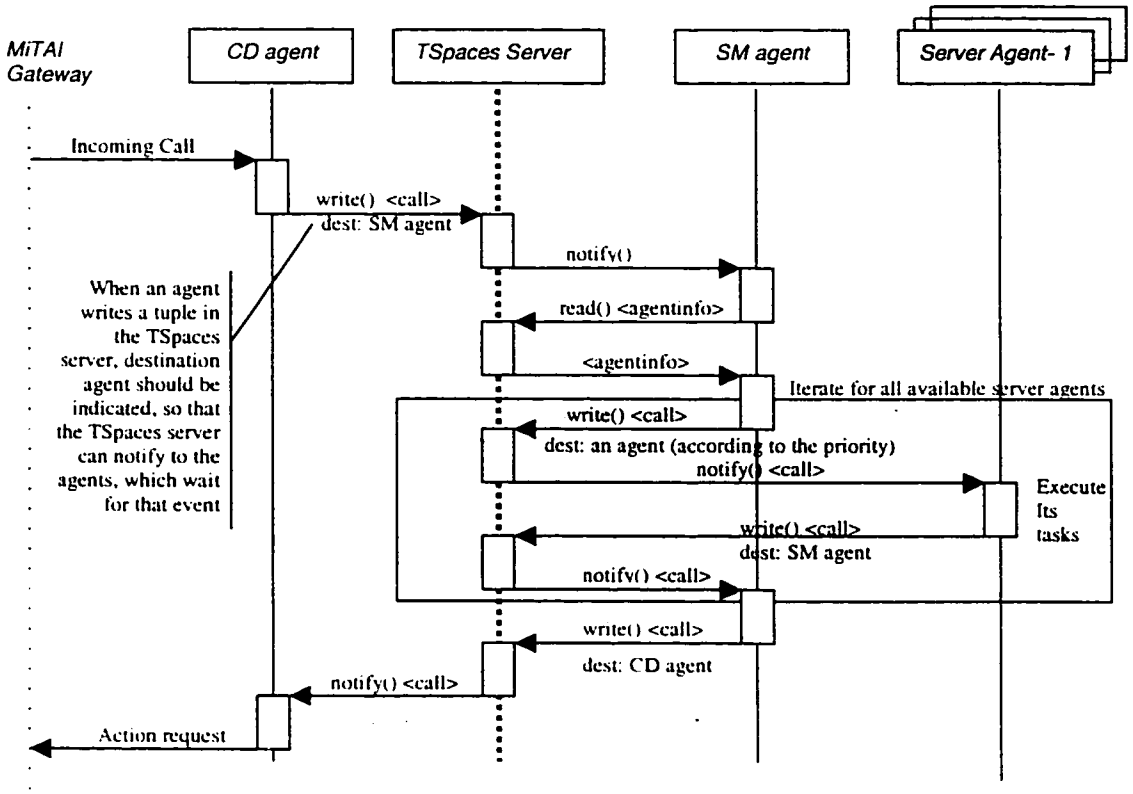


Figure 3-9: The Sequence Diagram for Call Processing

The server agents execute their task when they receive the request from the SM agent and then response to the SM agent with the result of the execution. When the server agent, which has the lowest priority, responses to the SM agent, the SM agent sends the <call> tuple to the CD agent. That <call> tuple includes the "action request" and the CD agent sends the "action request" to the MiTAI Gateway.

Chapter 4

Implementation and Testing of ACD System

The ACD system has been implemented using the Java programming language on a Windows NT platform. A number of packages have been used for the implementation:

- The Java 2 Platform, Standard Edition v1.3.1, for a Java development environment
- The TSpaces v2.1.2 as a data repository and a communication medium between agents
- Mitel Telephony Application Interface (MiTAI) for the PBX interface

The details of the system's installation are shown in the *Appendix I – System Installation Guide*.

The design of the ACD system is not restricted to any specific domain of users. A flexible method of defining knowledge for a user domain allows the system to be used in different domains. A system administrator can define a hierarchy of the user's location, activity and time according to a domain of target users. For testing purposes, the system provides two example domains: an office worker's domain and a professor's domain. A tester may choose one of the two domains by clicking the corresponding button in the introduction window. It sets up the necessary knowledge such as the hierarchy of possible locations, the relationship information and the buddy-list automatically.

4.1 Server

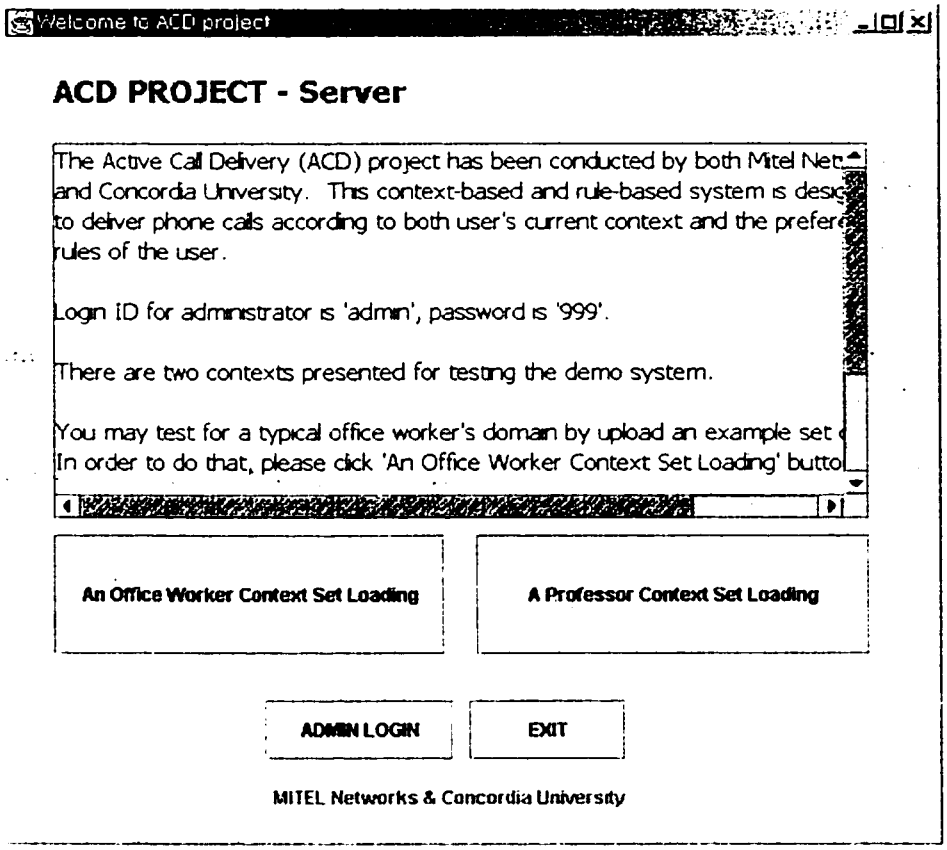


Figure 4-1: The Server Welcoming Window

The ACD server system was designed to be simple and easy to use. The installation procedure for the ACD server system requires unpacking the Java class files and executing them on any machine on the network.[9] At the start, a welcoming window provides brief information about the ACD project, administrator login information, and two buttons to load information for testing purposes: "An Office Worker Context Set Loading", and "A Professor Context Set Loading". When "An Office Worker Context Set Loading" button is clicked, example contexts for an office worker will be written into the TSpaces. This model of the hierarchy of location and activity is shown in figure 4-4. For

[9] Details of installation procedures are available in the Appendix: System Installation Guide

testing an example of the professor's domain, "A Professor Context Set Loading" can be selected. A tester can start the server without selecting a pre-defined set of information for testing a customized context. The server system informs the tester that a hierarchy of context should be either selected from the two choices or set manually when a tester skips information loading. "ADMIN LOGIN" and "EXIT" buttons are self-explanatory.

4.1.1 Server Login

To login as an administrator, who controls knowledge and services for the server, one should be authenticated through the login window for an administrator role as shown in figure 4-2. An error message window will be presented if a field is omitted or there is a mismatch between Admin ID and Password.

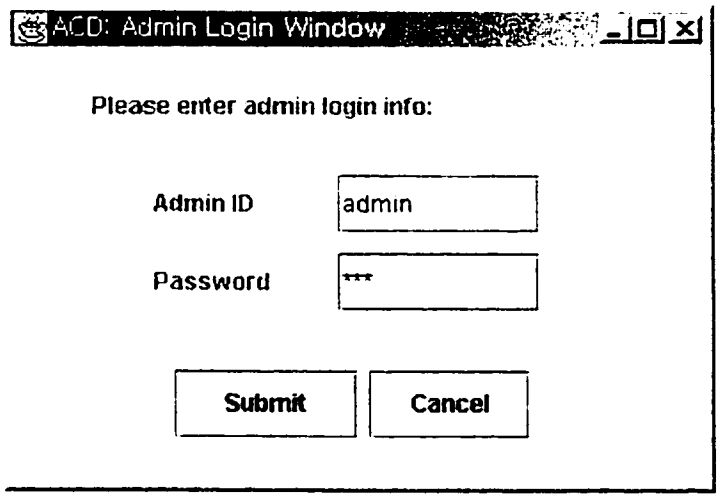


Figure 4-2: Logging into the Server

4.1.2 Server Main Window

If the login is authorized, the Server Main Window will be presented for further processing as shown in figure 4-3: System Knowledge Management, Startup Agents and Simulation.

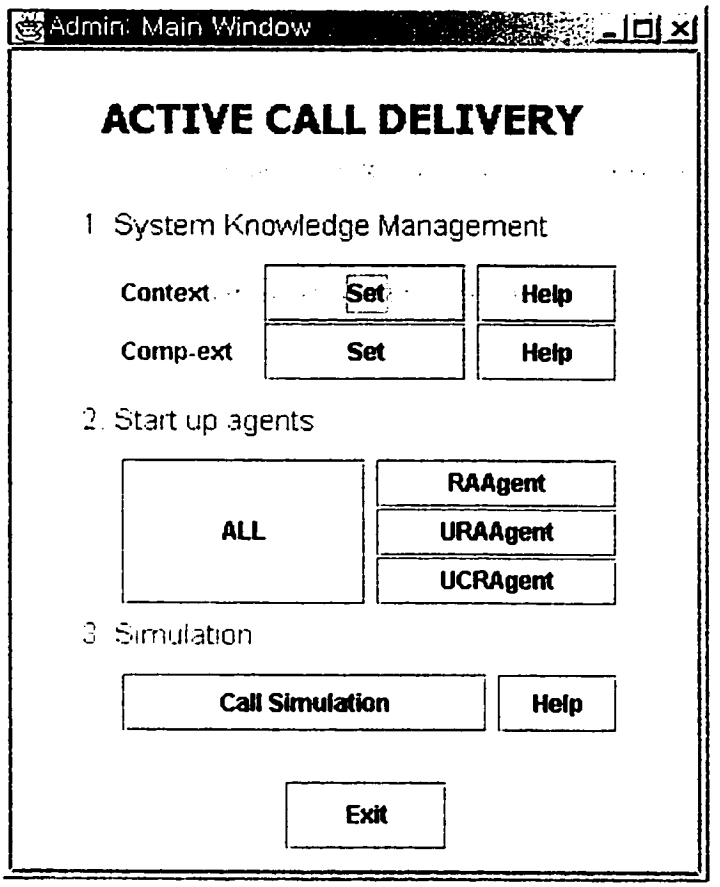


Figure 4-3: Server Main Window

4.1.2.1 System Knowledge Management

- Context Setting

A context hierarchy should be constructed before a client system provides user services. Clicking the "Set" button of a "Context" allows the administrator to set a hierarchy of contexts using the GUI. The predefined roots of the context hierarchy for

this system are *location* and *activity*. Time is another context being used in this system, but it is personalized based on a particular user's schedule. Therefore, each client system manages it's own time context. The context window with an example location hierarchy and activity is shown in figure 4-4. To add a new sub-context, click one of contexts in the hierarchy and click "Add" button. New child context will be added with the default name "New Node *n*". Click twice on the name to rename the context. To remove a node, click the node to be removed and click the "Remove" button. Clicking the "Clear" button will clear all the nodes from the context tree. To save changes and finish modifying, click "Finish".

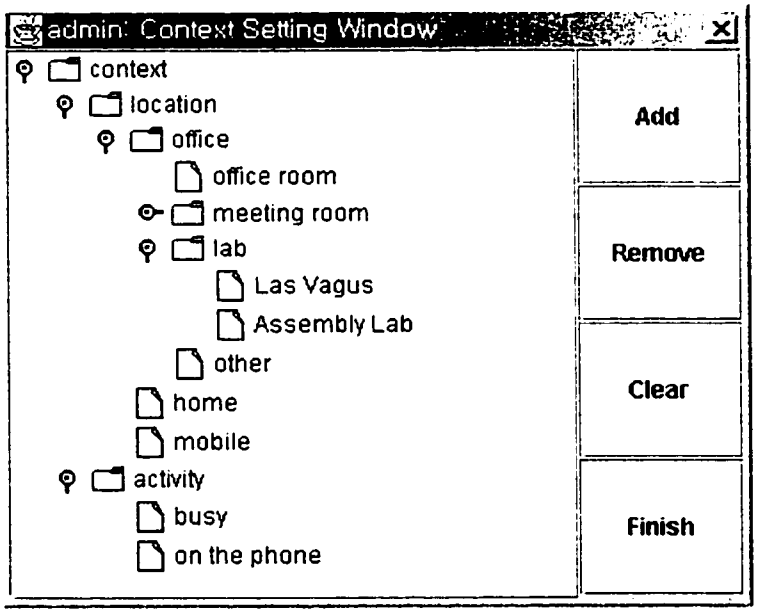


Figure 4-4: Context Setting Window

- Computer name and extension number setting

A computer name and a phone number are paired and saved in order to forward an incoming call. The information will be used by the system when the message delivery action is "Forward it to where I am" or when the user wants to forward an incoming

call to a different phone. An administrator can add, remove and change the information in the table shown in figure 4-5.

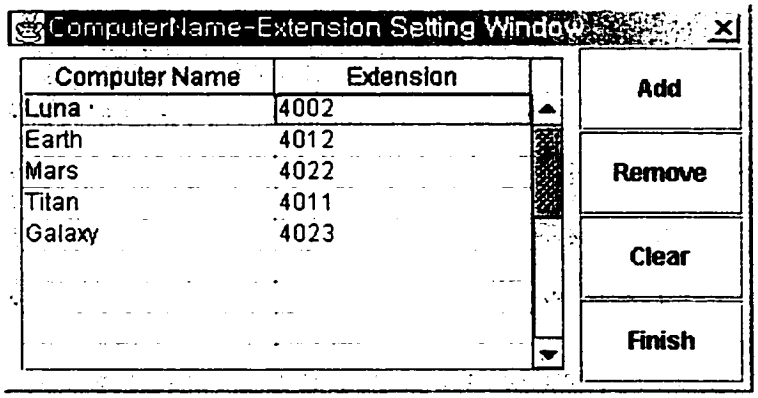


Figure 4-5: Computer Name and Extension Number Setting Window

4.1.2.2 Startup Agents

The server agents can be executed on any machine that has an access to the TSpaces server. This means that any machine within the network can be used to execute a server agent. This design gives a flexible distribution of agents. All the agents can be executed together on the current machine by clicking the "ALL" button, or each agent can be executed separately either on the same machine or on different machines within the network by clicking the corresponding button. Due to network constraints, each agent should report its status regularly. The reporting method is writing its status tuple every second, and the lifetime of the tuple is three seconds. The details of status management for the server agents are explained in the section 3.2.3. Each agent has a display window and four buttons to control it. Clicking the "Start" button starts a corresponding agent by activate its status report. The "Stop" button is for de-activating its status report for testing purpose. A maximum three seconds later, the status tuple for corresponding agent no longer exists in the TSpaces. The clients recognize that the agent is not available. The "Start" and "Stop" buttons are exclusive, in that one is

disabled when the other is executing. The "Clear" button clears the display area, and the "Finish" button terminates the corresponding agent.

The Relationship Assigning (RA) agent assigns relationship information between the caller and the receiver based on the user's buddy-list. An example of an execution is shown in figure 4-6. In the figure, a request for relationship-assignment from a System Management (SM) agent for a user, who has a user ID "choi", is received. This request comes with the caller's phone number, which is "4021" in this example. The RA agent retrieves the user's buddy-list from the TSpaces and finds a relationship between a user "choi" and a person who has the phone number "4021". As a result, a "boss" relationship is identified and the call control is sent back to the client by writing a tuple with relationship information into the TSpaces.

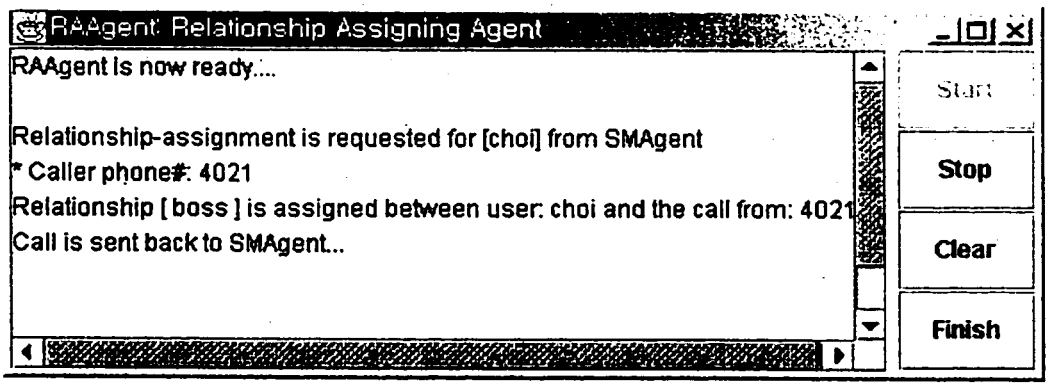


Figure 4-6: Relationship Assigning Agent Window

The User Rule Assigning (URA) agent assigns all user preference rules that matched the conditions of the rules and the user's current context. If the condition of a rule is made of information that has a hierarchy, sub-categories will be examined. For example, a location condition of a user preference rule is 'If I am in the *office*'. Sub-locations of the *office* such as *lab*, *meeting room* should also satisfy the rule's condition.

Suppose that the user "choi" has a call from extension number "4021" while he is in the *meeting room*, and he is *busy*. In the testing scenario, "4021" is *Thomas Ragan*'s phone and he is the user's *boss*. The matching user preference rules based on his current context, relationship information and the caller are as follows:

Rule name: Worker-office rule
Condition: If a call is coming from [*worker*] relationship
And when I am in my [*office*]
Action: Put through the call

Rule name: Thomas Ragan-busy rule
Condition: If a call is coming from [*Thomas Ragan*]
And if I am [*busy*]
Action: Ask the caller what to do

Rule name: worker-office-busy rule
Condition: if a call is coming from [*worker*] relationship
When I am in my [*office*]
If I'm [*busy*]
Action: Forward it to voice mailbox

The names of the rules that satisfy the conditions are displayed as they are matched in figure 4-7. Although these rules are satisfactory for the user's current context, the system needs to select one rule that is most appropriate for the user in order to take an action.

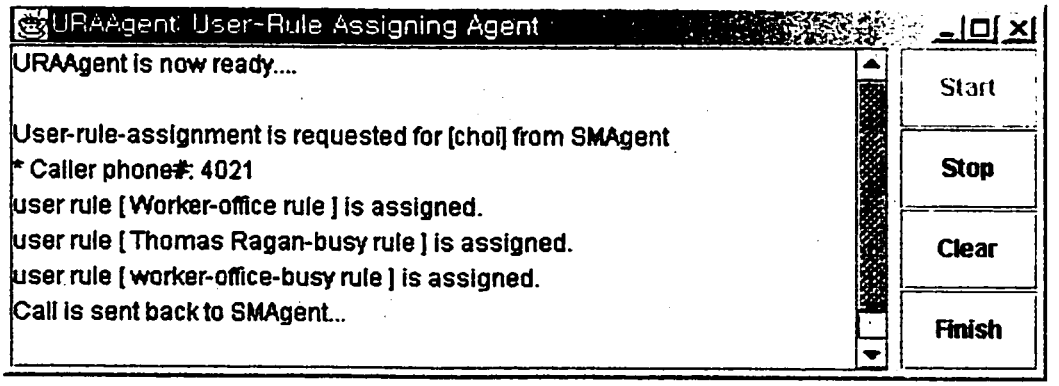


Figure 4-7: User Rule Assigning Agent Window

The User-rule Conflict Resolving (UCR) agent selects one user preference rule if there are more than one rules assigned by the URA agent. It selects the most specific among the assigned rules. A rule with more conditions is considered more specific. In the testing scenario, the "worker-office-busy rule" is the most specific rule among the assigned rules, so it should be selected as shown in the figure 4-8.

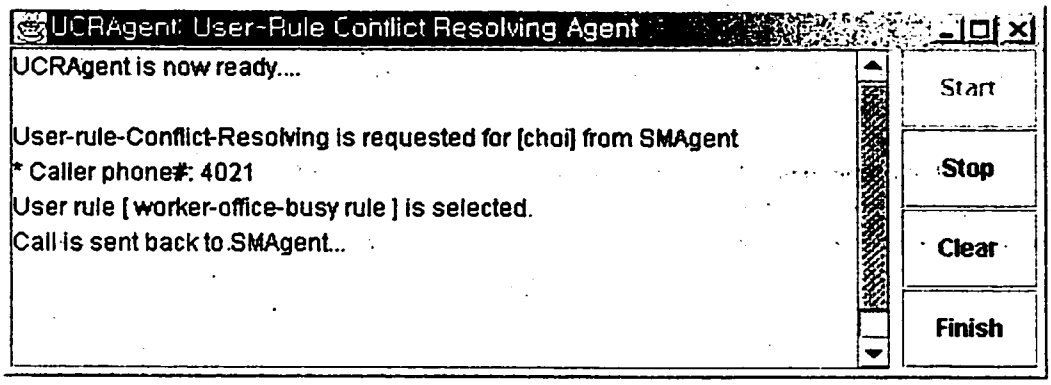


Figure 4-8: User-rule Conflict Resolving Agent Window

4.1.2.3 Call Simulation

Call Simulation is designed for testing and simulating call processing without connecting to phone sets. It runs on a server machine, so that the receiver's user id and phone number are necessary inputs as well as the caller's phone number. Clicking the "Call" button makes a simulated phone call as if the system receives an actual phone call.

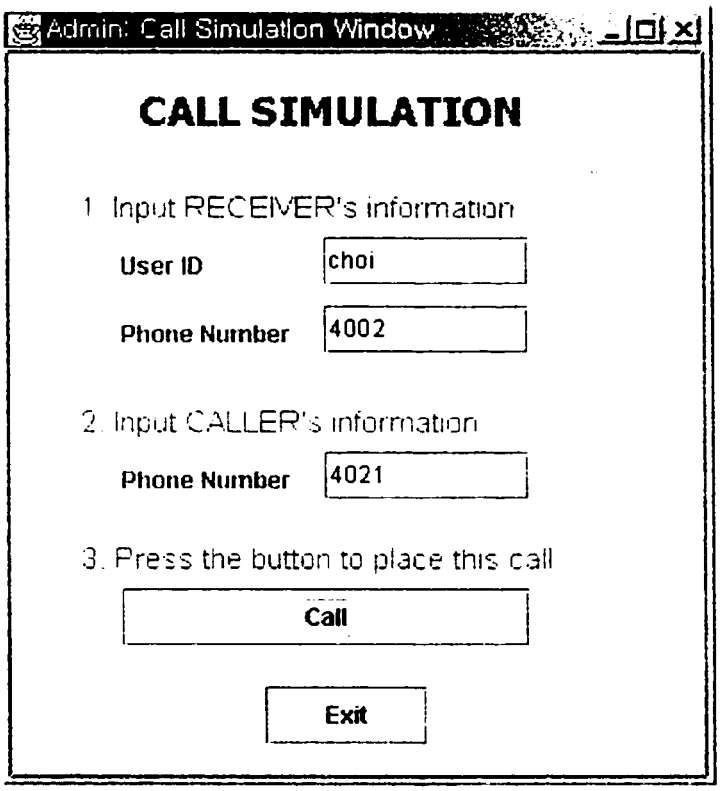


Figure 4-9: Call Simulation Window

4.2 Client

When the ACD client starts, the user is presented with a welcoming window as shown in figure 4-10. A brief explanation of the project and the testing information are explained on the white text area. Two buttons, the "An office worker info loading" button and the "A professor info loading" button, are used for testing each domain of users. All the necessary personal information for testing the client (user ID, password, user name, phone numbers, hierarchy of personal relationships, the buddy-list table, the time schedule, and user preference rules) will be copied into the TSpaces upon clicking on the button. A confirmation window will show feedback of the result of the processing. A tester can start the client system without selecting a pre-defined set of information for testing. The client system informs the tester that all the necessary user information should be either selected from two choices or set manually for all the necessary information if the tester skips the information loading.

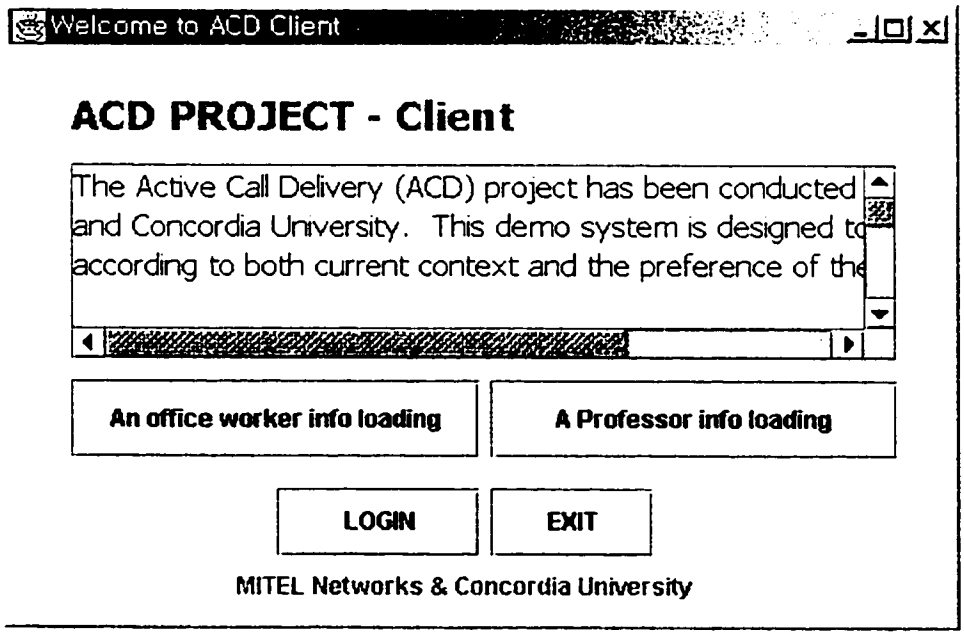


Figure 4-10: Client Welcoming Window

A user or a tester must authenticate himself at the beginning of each session, either by login or by registration, before being allowed to use this system. The "LOGIN" button will remove the welcoming window and pop up a new window for the login. The "EXIT" button causes shutdown of the client system. A warning window will confirm this action.

4.2.1 Client Login

During the login process, the user's ID and password will be checked with the information on the TSpaces server. If the user is new to the system, he has to register through the "Register" option. Validation and verification for each input field will be performed on clicking the "Register" button in registration window. Once a user is registered and logged-in, the user name appears on every client window frame as a feedback about the user's identification.

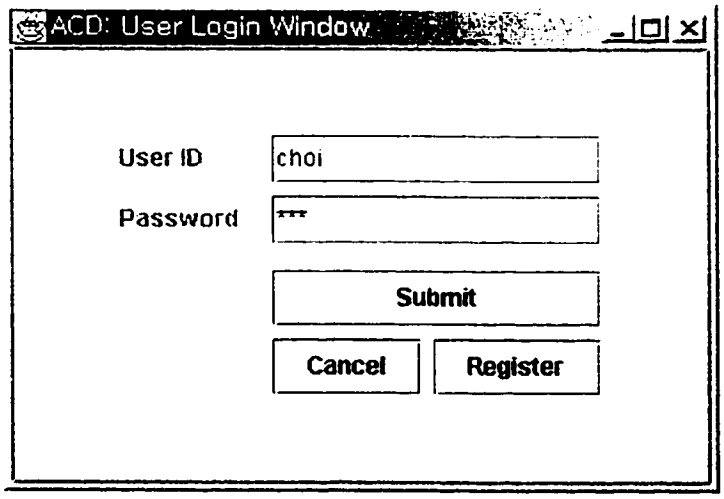


Figure 4-11: Logging into the Client

4.2.2 User Registration

A first-time user should provide basic user information such as user ID, password, name and phone numbers through registration. The input user ID will be checked to see if it is not duplicated with existing user IDs. Each field has its own constraints of length and format. An error window notifies the user if any constraint is violated on clicking the "Register" button in figure 4-12.

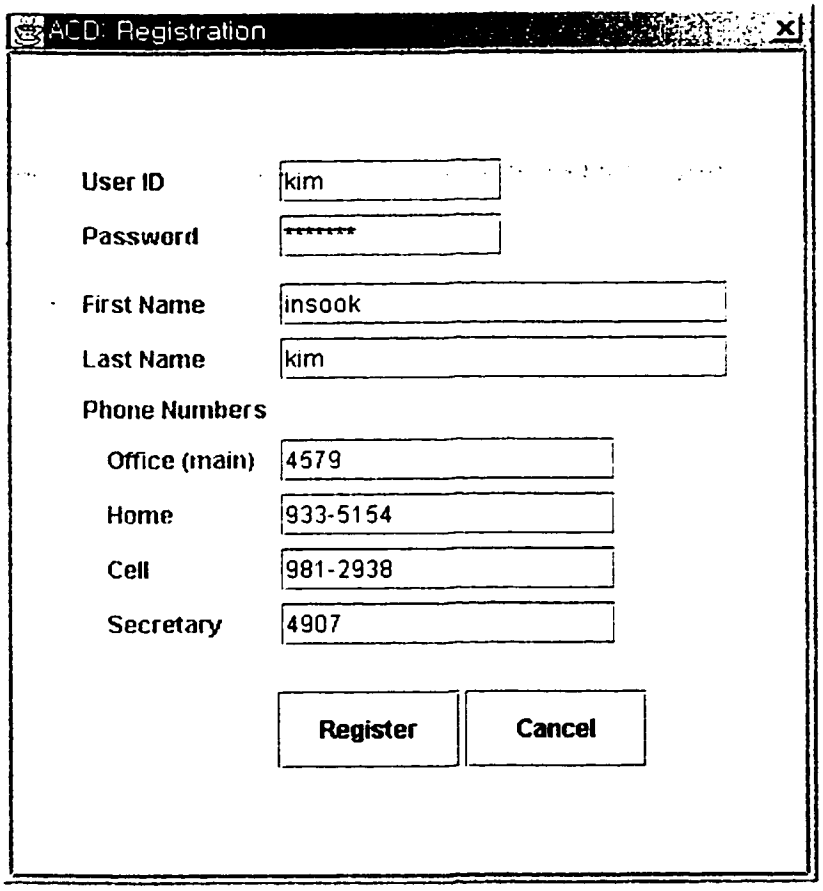


Figure 4-12: Registration Window

4.2.3 Client Main Window

The main window for client control will be presented if login or registration process is successful. It consists of three parts: User Information, Knowledge Management, and Context Simulation.

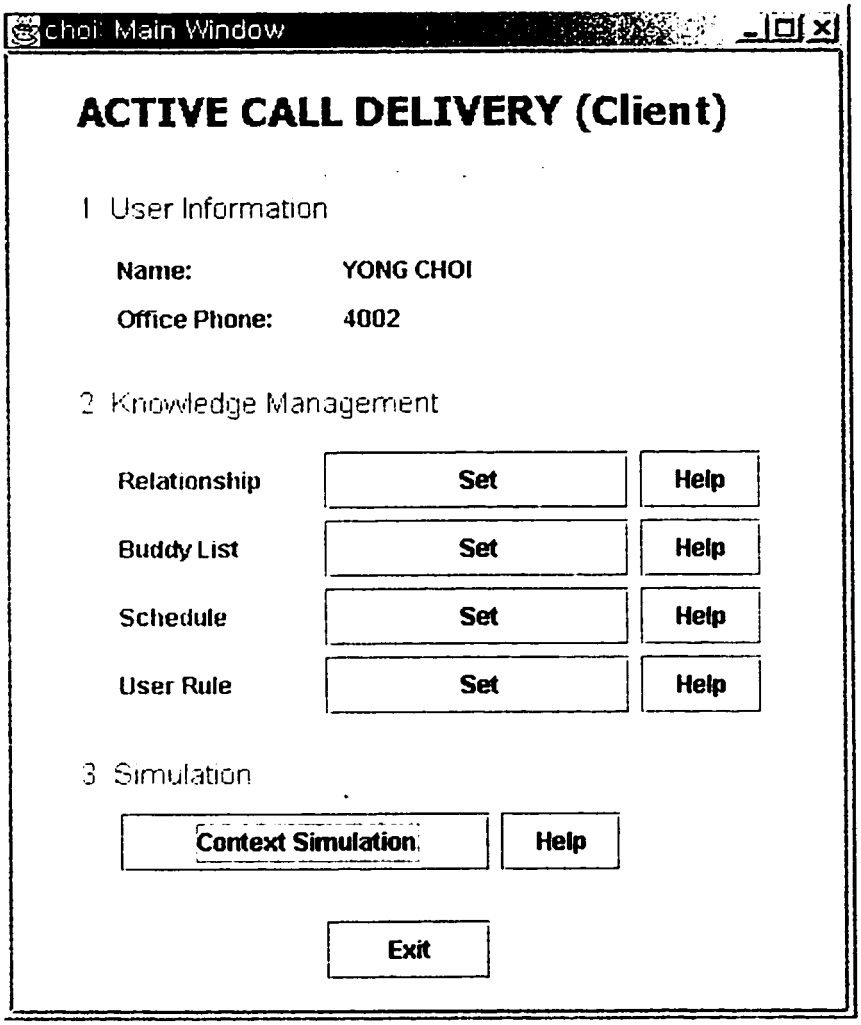


Figure 4-13: Client Main Window

4.2.3.1 User Information

The basic user information is displayed as a feedback; the user name and the office phone number. The user ID is displayed in the frame of the window.

4.2.3.2 Knowledge Management

The user can set his personal information such as relationship information, the buddy list, the schedule and the user preference rule through this menu. Each menu has a help button to give a brief explanation of the corresponding item's function.

- Relationship Setting

The personal relationship information is shown as a tree structure that is easy to maintain (figure 4-14). To add a new sub-relationship, select one of relationship nodes and click the "Add" button. A new child node will be created with a default name, "New Node *n*". Click twice on the name to rename the relationship. To remove a relationship, select the removing relationship nodes and click the "Remove" button. Note that sub-relationships that belong to a deleted relationship will also be removed. To remove all of the relationships, click the "Clear" button and it will clear all the relationship nodes from the tree. To save changes and finish modifying, click "Finish".

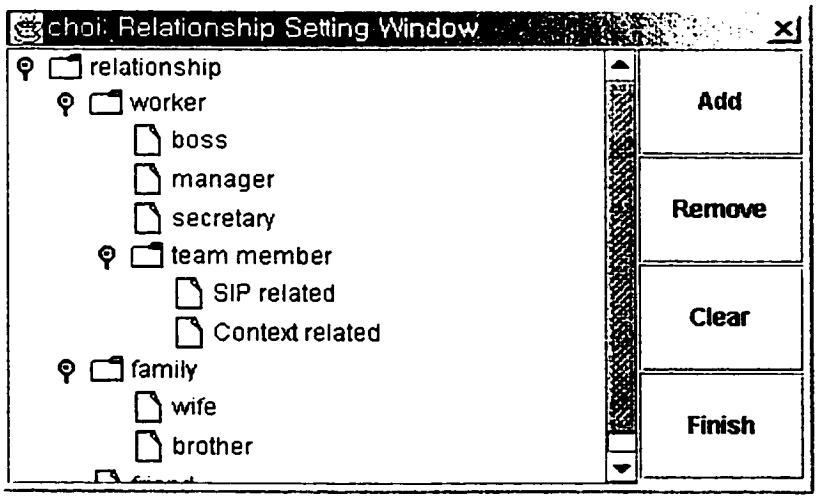


Figure 4-14: Relationship Setting Window

- Buddy-list Setting

Clicking any of the fields on the table in figure 4-15 allows the user to make changes to that particular field. To remove a set of buddy information from the table, select a column and click "Remove". The modified table will be saved into the TSpaces when the "Finish" button is clicked.

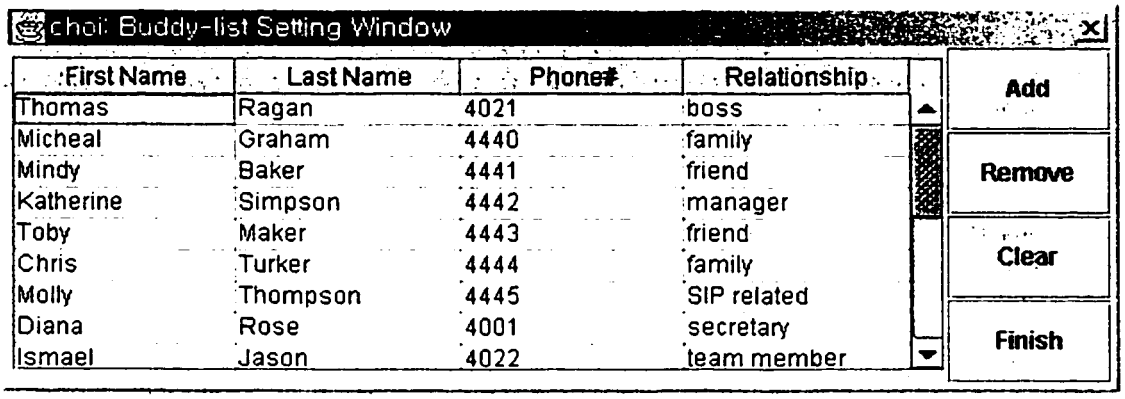


Figure 4-15: Buddy-list Setting Window

- Schedule Setting

A user can set two categorized schedules: lunch time and meeting time. When a user creates a new preference rule, these time settings can be referenced as "lunch time" and "meeting time". Select the start time and the end time for each schedule from the pull-down menu in figure 4-16. The "Finish" button saves the schedules and removes the Schedule Setting Window.

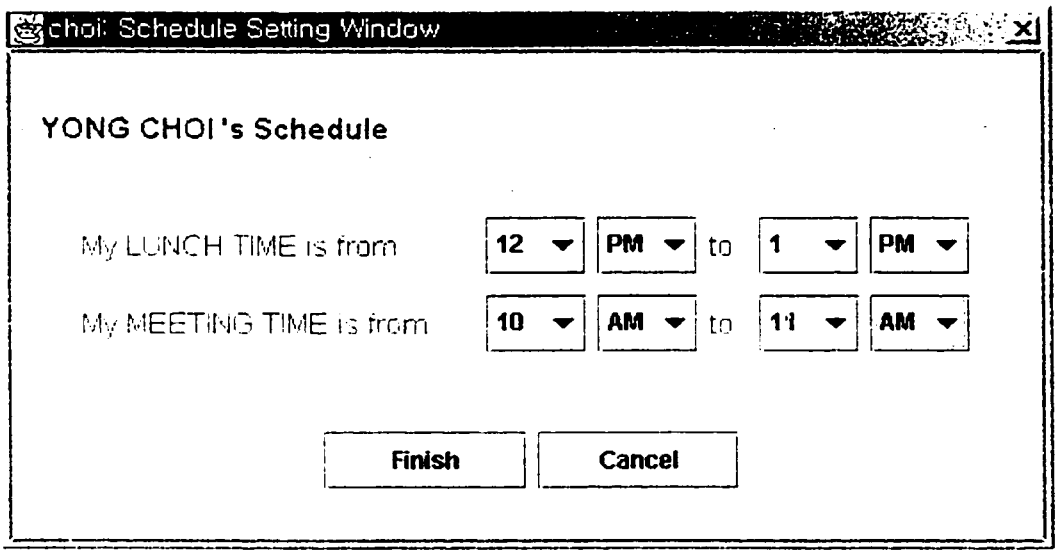


Figure 4-16: Schedule Setting Window

- User-rule Setting

The user-rule Setting Window consists of three parts: the user rule table, which consists of sequence numbers and user rule names, UI buttons, and a Description window, as shown in figure 4-17. Clicking on one of the rules in the table allows the user to see the description of the selected rule in the Description window. The buttons such as add, refresh, remove, clear and finish are used for managing rules. The "Add" button is designed for creating a new rule and it takes four steps, which are explained in later paragraphs. Clicking the "Refresh" button makes newly created rule showing

in the user rule table. To remove an existing rule, select a deleting rule on the table and click the "Remove" button. To remove all the existing rules, click the "Clear" button. Note that it does not warn a user before deleting everything. To finish editing, click the "Finish" button and save any changes.

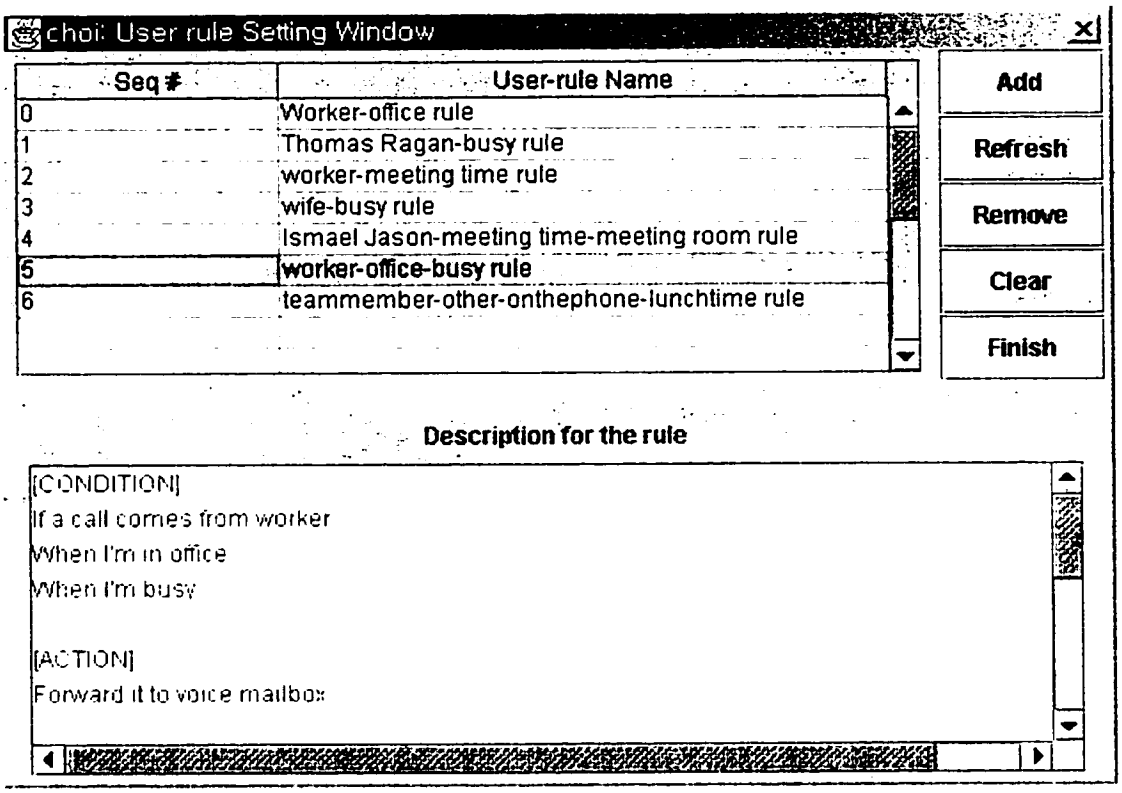


Figure 4-17: User-rule Setting Window

Clicking the "Add" button in the "User-rule Setting Window" will start creating a new rule. Adding a new user rule takes four steps. The first step is selecting the contexts as a part of the conditions of the rule being created (figure 4-18). The location and activity selection can be made from a given hierarchy tree. These hierarchies of location and activity are defined by an administrator from a server. The time context can be selected from a pull-down menu with three choices: "any time", "meeting time" and "lunch time". The actual time schedules are set by the user through the "Schedule Setting Window." The steps are displayed at the bottom of the window and the current step is shown in red. When the context conditions have been selected, click the "Next" button to move to the second step.

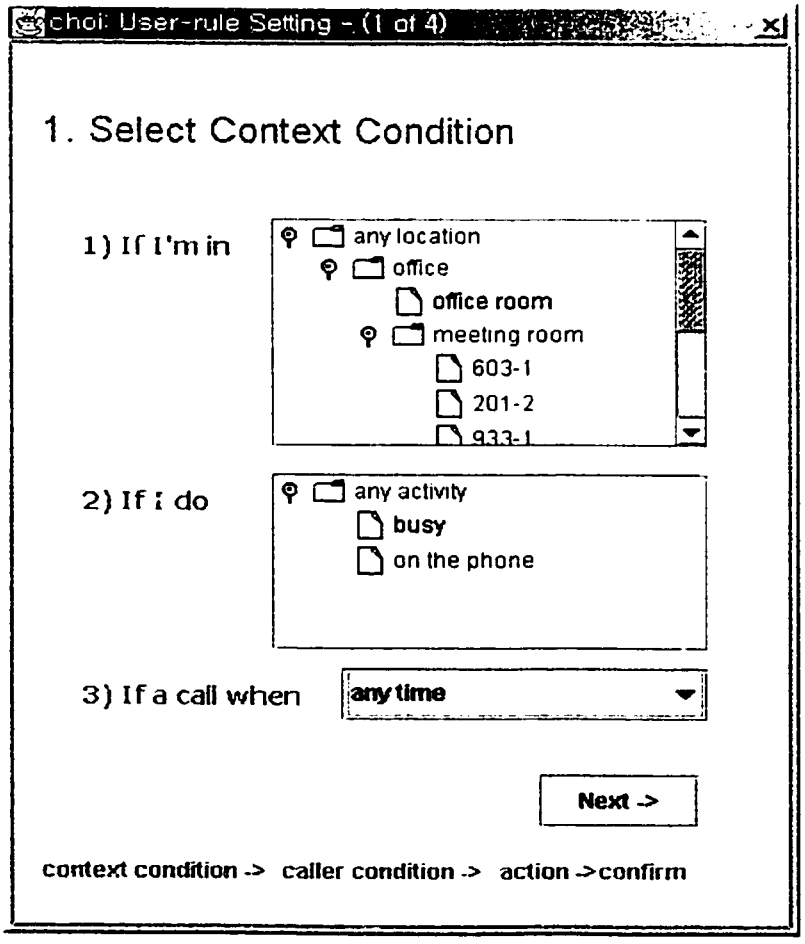


Figure 4-18: Creating a New Rule – 1. Select Context Condition

The second step is selecting a caller-type as a part of the condition. One of three categories can be selected: any caller, a buddy list table, and a relationship tree. These three categories are exclusive, so that radio buttons are provided to select only one category. When this category is selected, a user can then select items in its selection window. Figure 4-19 shows an example of the selection of "Buddy". One of the buddies can now be selected from the buddy table, whereas the relationship hierarchy window remains disabled.

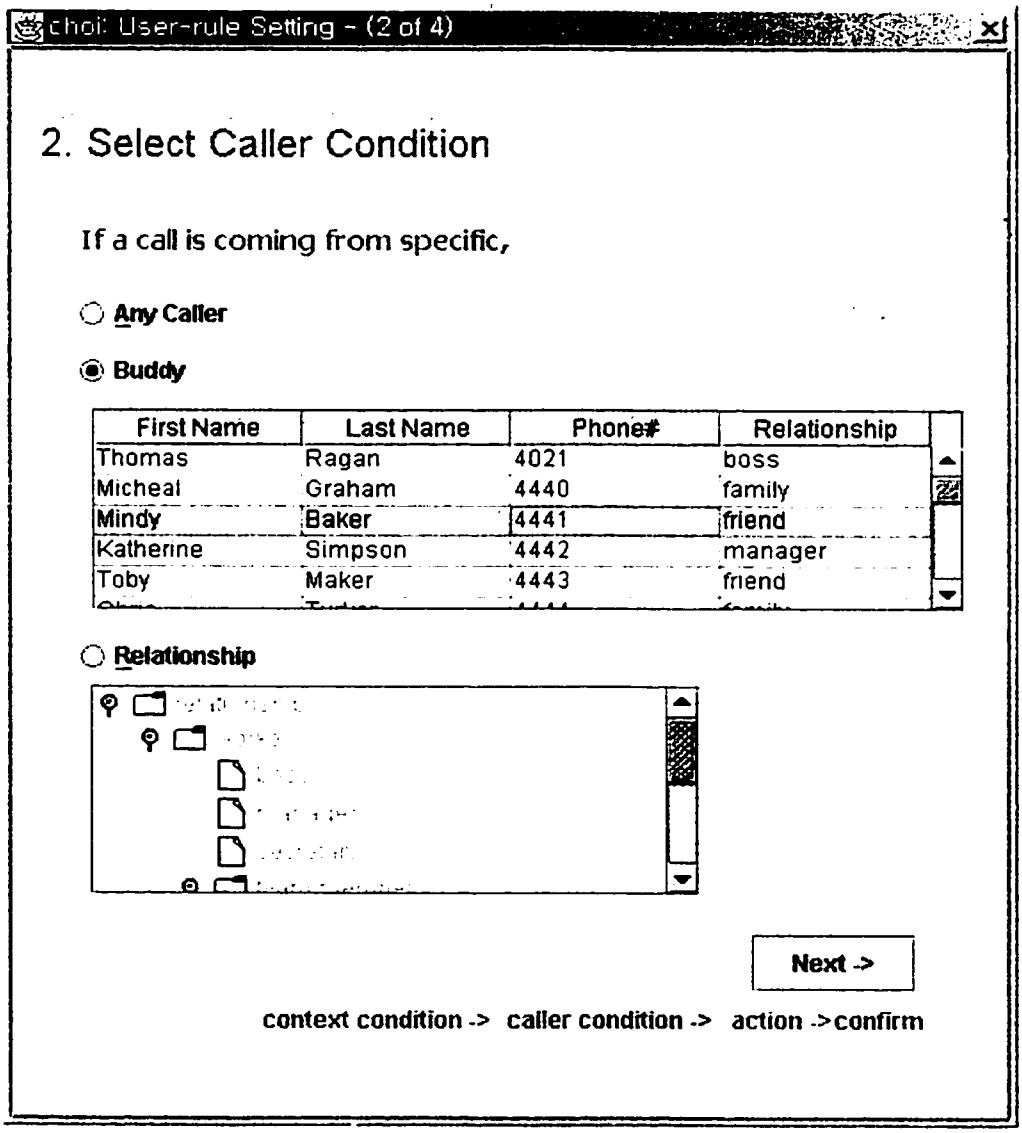


Figure 4-19: Creating a New Rule – 2. Select Caller Condition

The third step is selecting an action of the rule from the predefined list of actions. The action items are listed with their associated radio buttons, and only one can be selected from the list. "Forward it to voice mailbox" and "Ask to the caller what to do" actions have not been implemented due to implementator's time constraints.

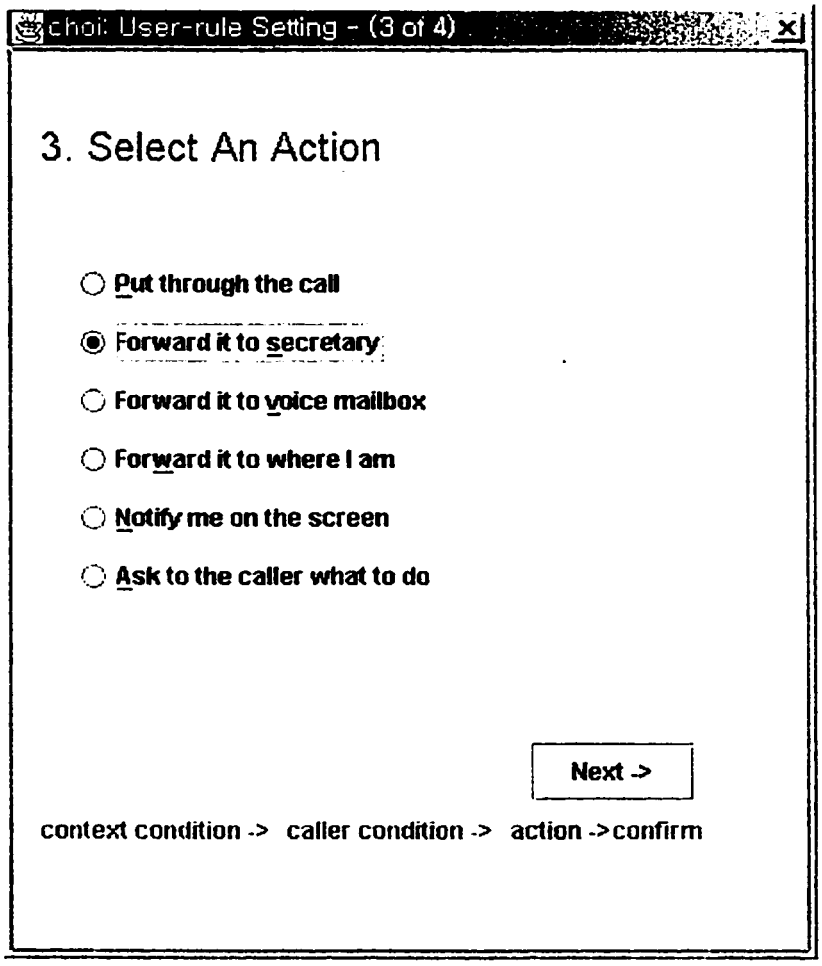


Figure 4-20: Creating a New Rule – 3. Select An Action

The fourth and the last step for creating a new rule is confirmation. A user should confirm and assign a rule name. The rule name should be unique. The "Description for the rule" window shows the selections the user made: the condition(s) and an action. Clicking the 'Submit' button saves the new rule.

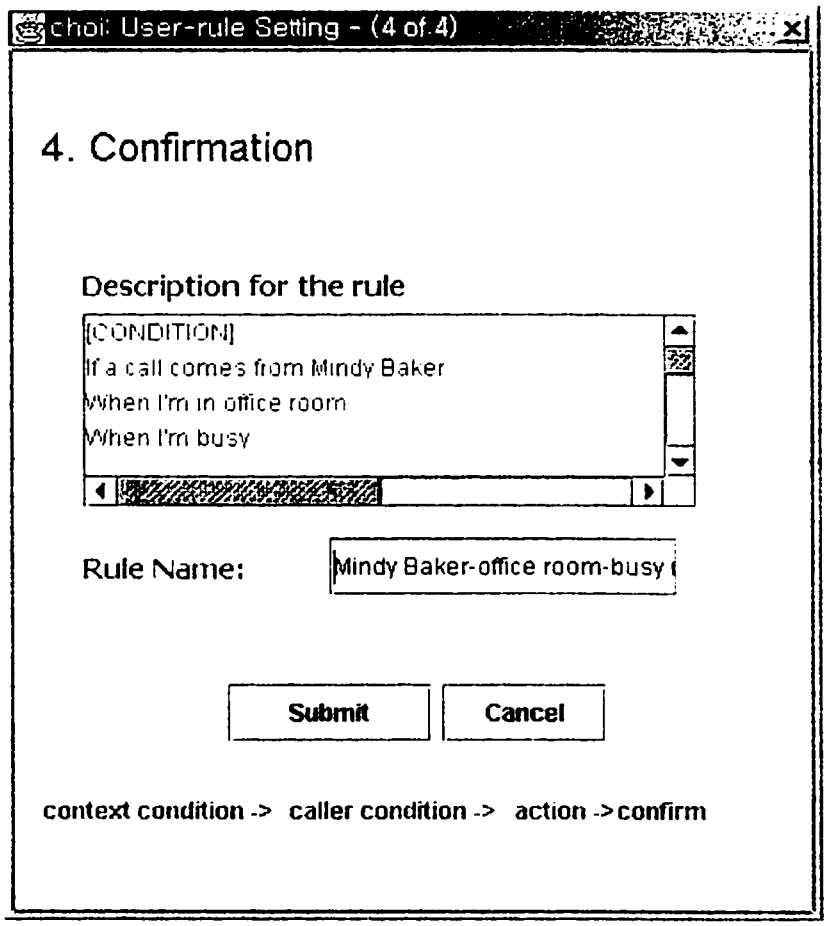


Figure 4-21: Creating a New Rule – 4. Confirmation

The description of the created rule shown from figure 4-18 to figure 4-21 is as follows:

Rule name: Mindy Baker-office room-busy rule
Condition: If a call is coming from [*Mindy Baker*]
And when I'm in my [*office room*]
And when I'm [*busy*]
Action: Forward it to secretary

The entire process to create a user preference rule is shown in figure 4-22. The first two steps are for selecting conditions of a creating rule, so that the user can skip either the first or the second step according to the creating rule.

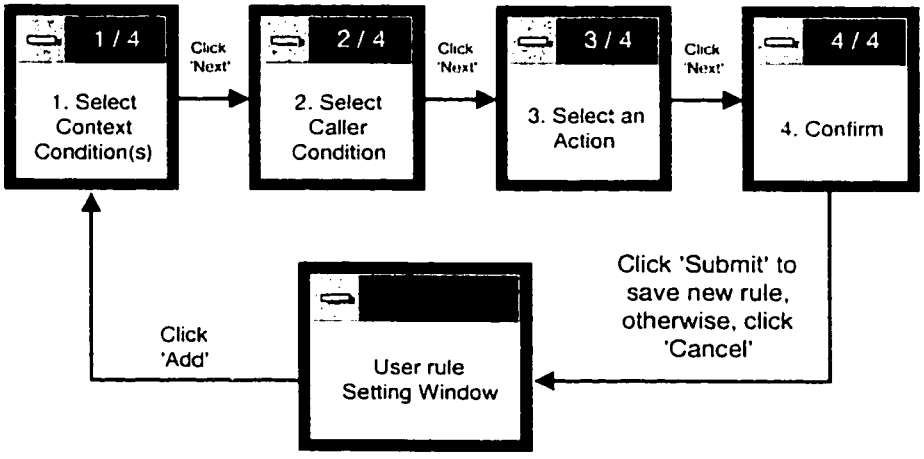


Figure 4-22: Creating a New Rule – Entire Process Diagram

- Context Simulation

Ultimately, the user's current contexts such as the current location and activity should be updated by the Context Agent. In our test case, a simulation program substitutes the occurrence of events in a real-life. For testing purposes, the tester selects one of the desired contexts on the hierarchy trees, and then clicks the "Apply" (figure 4-23). The current time, which is displayed on the window, is the client machine's system time that is used as time context by matching with a user's schedule.

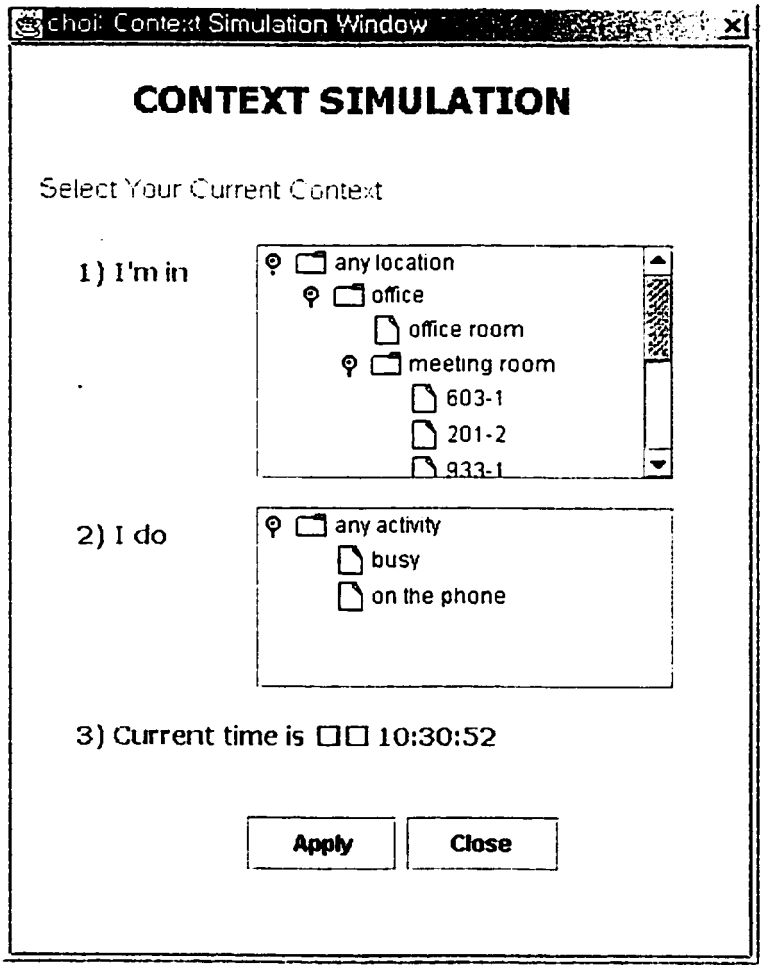


Figure 4-23: Context Simulation Window

4.2.4 Client Agents

The client has two agents: the Call Delivery (CD) Agent and the System Management (SM) Agent. Each agent has its own display window to present procedural messages to a user. When the CD agent gets started, it should confirm the necessary connections. The CD agent should be connected to both the TSpaces server to communicate with other agents and to the MiTAI Gateway server to communicate with a phone system.

The window in figure 4-24 displays the machine name and the port number of the TSpaces server, which this client is connected to. The default TSpaces server name is "localhost", which is the same machine as the current client machine. The second line shows the MiTAI Gateway server name and its port number. The "CDAgent for [choi] is now ready" means that the two necessary connections are confirmed, and the CD agent is ready for the user, whose user ID is "choi".

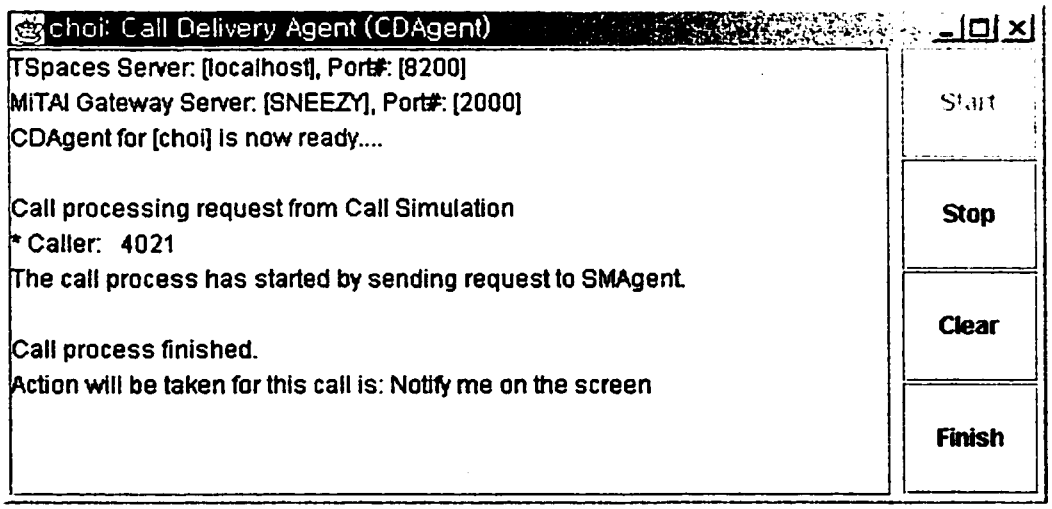


Figure 4-24: Call Delivery Agent Window

A call processing request can be received from either the Call Simulation or the Call Monitor. The Call Monitor communicates with the MiTAI Gateway server for handling actual phone calls whereas the Call Simulation is another window on a server machine to

test the system without MiTAI Gateway interfaces. When call processing has finished involving all the available agents, the CD agent extracts the selected user rule, which is a result of the processing, and requests the Call Monitor to carry out the action stated in the selected rule. When the example in figure 4-24 is executed, the action, "Notify me on the screen", will show a notification window on the client machine. The window might look as shown in figure 4-25.

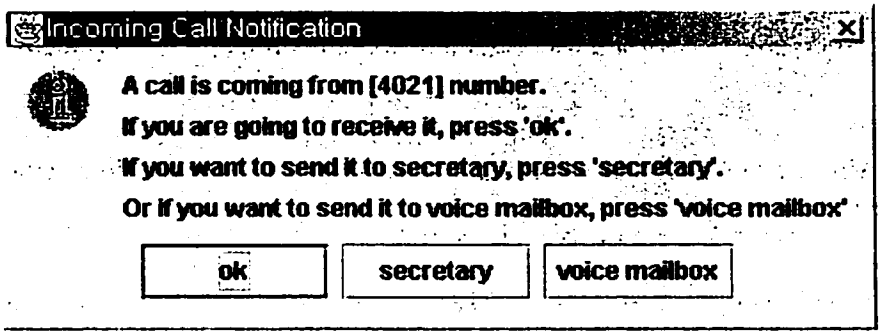


Figure 4-25: Incoming Call Notification Window

The SM agent should be connected to the TSpaces server to communicate with other agents. The display in figure 4-26 confirms the established connection. The default TSpaces server name is "localhost", which is the same as the CD agent's default server name. "SMAgent is for [choi] now ready" means that the necessary connection is confirmed and the SM agent is ready for the user, whose user ID is "choi". The SM agent is responsible for sequencing the available agents according to their priority. The display window shows the sequencing of the agents as a part of the call processing. When the CD agent notifies the user about an incoming call, the SM agent retrieves the status of the agents and distributes a call control to the each agent. On the completion of the call processing, the control will be sent back to the CD agent to execute the selected action. The SM agent window has an "Agent Status" button which allows the user to check the

agent status manually. The "Clear" button clears the message display area and the "Finish" button exits the system.

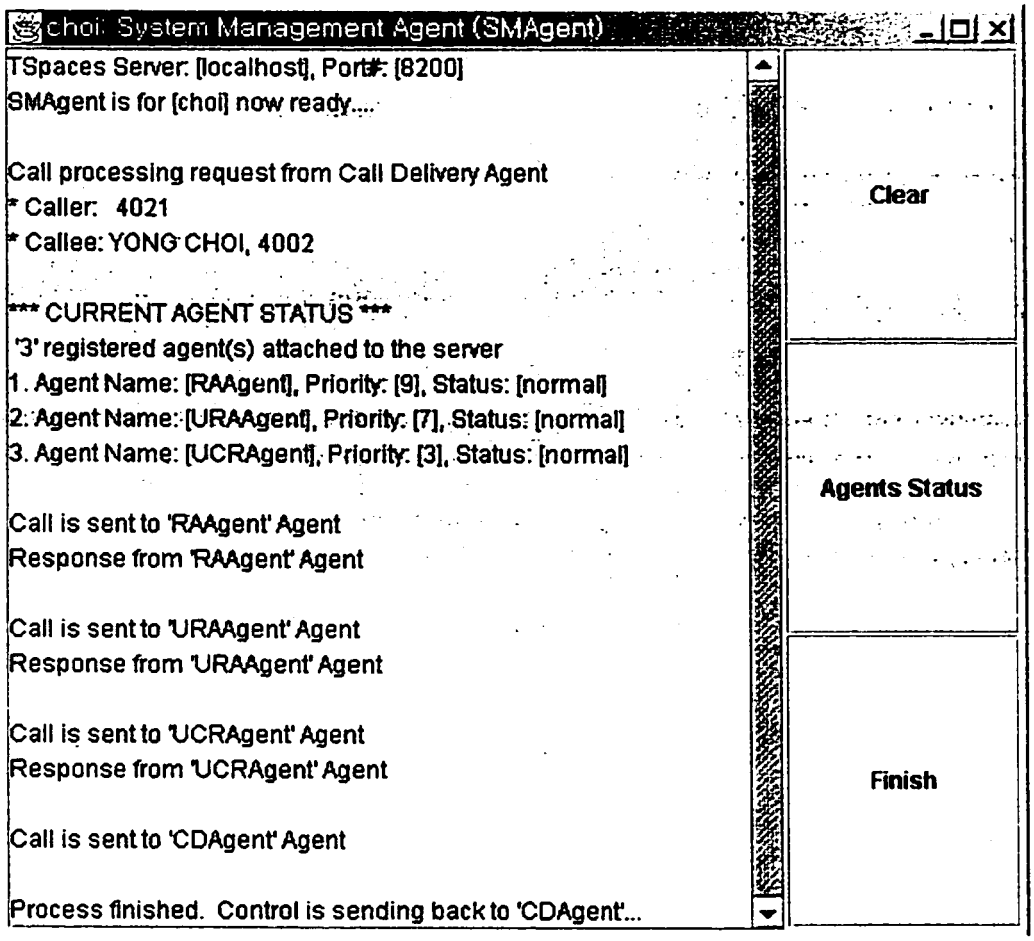


Figure 4-26: System Management Agent Window

Chapter 5

Related Research and Systems Dealing with Location

Among the different parameters used in defining a context, the "location" parameter has been most actively pursued by researchers to enhance the usability of application systems. The technology for location-based services is growing. Several industry based interoperability forums have been formed and they cooperate in the development of location-based technology standards. Existing sample location-based research and commercial systems that differ in their respective architectures and designs are introduced in this chapter. As a context-based message delivery system, our prototype system can make use of some of these technological approaches to detect user's location and track the dynamic changes in location. There are issues to be solved such as the "Big Brother" factor, criminal misuse, and obtrusive advertising, but these issues are beyond the scope of our thesis.

5.1 Active Badge System

The Active Badge System (ABS) [Want, 92] determines the location of an individual automatically by using an Active Badge and forwards the incoming call based on that information. An Active Badge emits a unique code, called a beacon, for approximately a tenth of a second every 15 seconds. These periodic signals are picked up by a network of sensors placed around the host building. A master station, also connected to the network, polls the sensors for badge 'sightings', processes the data, and then makes it available to clients that may display it in a useful visual form.

The badge is designed in a package roughly 55x55x7mm and weights a comfortable 40g. Pulse-width modulated infrared (IR) signals are used for signaling between the badge and the sensor: IR solid-state emitters and detectors can be made very small and can be very inexpensive (unlike ultrasonic transducers) and they can be made to operate with a 6m range. The signals are reflected by partitions and therefore are not directional when used inside a small room. Moreover, the signals will not travel through walls, unlike radio signals that can penetrate the partitions found in office buildings. Because IR technology has already been exploited commercially, it is inexpensive and readily available for developing new applications.

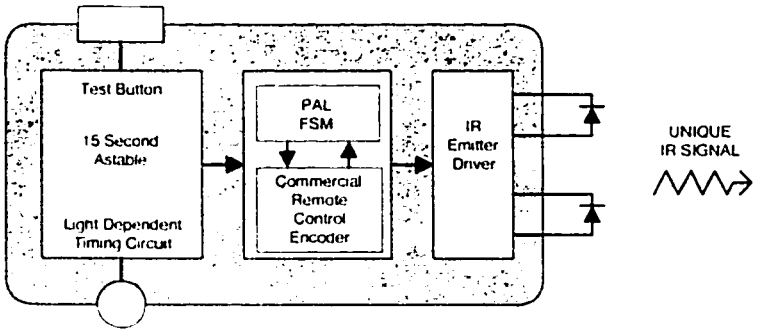


Figure 5.1: The Design of an Active Badge

A prerequisite for the badge network was that it should be able to link all areas of any building with an arbitrary topology. Power would need to be fed though the network because the sensors would be too numerous and distributed in too many remote places for them to be supplied by power locally. The data-transfer format is logically the same as RS232, but the network is physically a wired-OR system. The consequence is that by using a simple level-shifting interface-box, any computer with a RS232 port can be used as the network master. The design of the network, shown in the figure 5.2, was conceived that would allow an independent network to support up to 128 sensors, controlled from the RS232 port of any standard workstation. This approach allowed a network of

workstations, joined by an Ethernet, to support multiple badge networks, the data being relayed back to one master server by conventional network protocols.

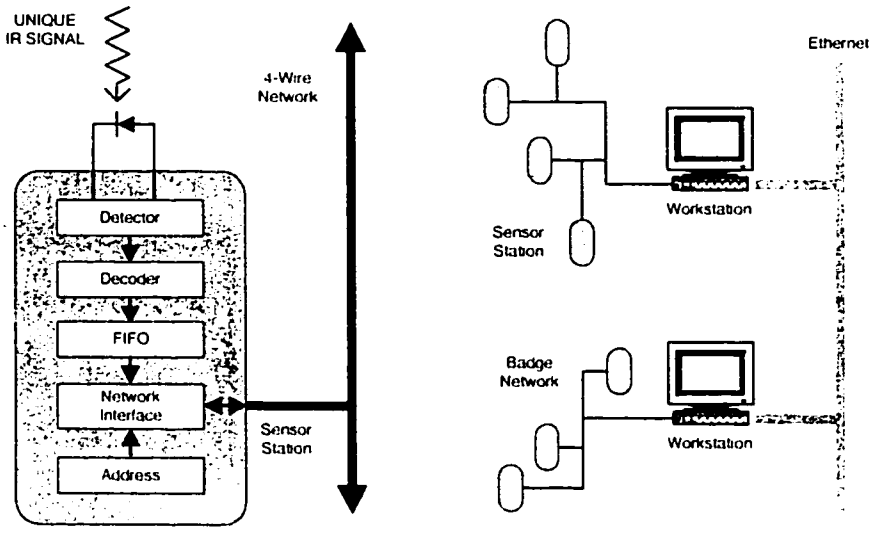


Figure 5.2: A Badge Sensor and Telemetry Network

The initial application system based on an Active Badge and a network was intended to be an aid for a telephone receptionist. Sensors were mounted in offices, common areas and major corridors. The system provides a table of names against a dynamically updating field containing the nearest telephone extension and a description of that location. The system also provides the likelihood of finding somebody at that location. If the chance of finding a person at a location is below 100%, that indicates the person is moving around, and if they have not been located for 5 minutes, the field contains the last time and location at which they were sighted. The last sighted location is still the best clue a receptionist may have to locate somebody and indeed there may be other colleagues in that area who will know why that person is no longer there.

People have found that a PBX controlled ABS with the option of human intervention to be of considerable use to office support employees. Although privacy is still a

controversial issue, this concern considerably diminishes after users have used the badges for a period of time.

5.2 Locust Swarm

The Locust Swarm system [Kirsch, 97] is a more enhanced system than Active Badge System in the aspect of privacy concerns and environmental issues. The Locust Swarm system provides inexpensive messaging and location information without batteries and without its own network whereas the Active Badge requires them. Furthermore, the Active Badges continually announce their presence to the environment which then, through a wired network, reports the location of the badge to a central system. Such centralized systems may fail to detect a user and are vulnerable to physical, social, and traffic analysis attacks. The Locust Swarm has been designed so that such attacks would require the physical presence of the "cracker" at each unit, or the installation of a parallel system equivalent in cost and labor to the Locust Swarm, and the cooperation of the user himself. The system is "privacy aware" in that it supplies information to the user's wearable computer and then the user can control how much of this information is shared with others or the installed infrastructure.

The unit of the Locust Swarm system for communicating with a user's wearable computer is called a Locust. Each Locust consists of a 4MHz PIC 16C84 micro-controller, a RS232 line voltage converter, infrared receiver, infrared LED, 6"x6" 9V solar cell, and a voltage regulator. The details of the design, instructions, parts lists, and codes are developed by the MIT Media Lab[10]. On power up the Locust begins broadcasting its location information as specified. A user's system can listen to this announcement and determine the user's location. The Locust also monitors the area with its infrared receiver to determine if a user is trying to upload an annotation about the area. By combining the

[10] Robert Poor, http://web.media.mit.edu/~r/projects/picsem/irx2_1/

abilities of Locusts with an appropriately equipped wearable computer, the user can interact with web-like hyperlinks, graphics, and sounds that are associated with objects in the physical world. In addition, the user can annotate and change these links as desired.

5.3 Hanging Messages

The "Hanging Messages" system [Chang, 01] is a messaging system that uses both location and time context, allowing users to specify time and place for message delivery. By incorporating delivery time and location, Hanging Messages allow users to send themselves reminders that will be triggered at appropriate times and places. For example, a user could send himself a shopping list, specifying the grocery store as the place and a particular day as the time parameter. This message would not only remind the user to go grocery shopping when he passed by the store, but would present him with his previously constructed list.

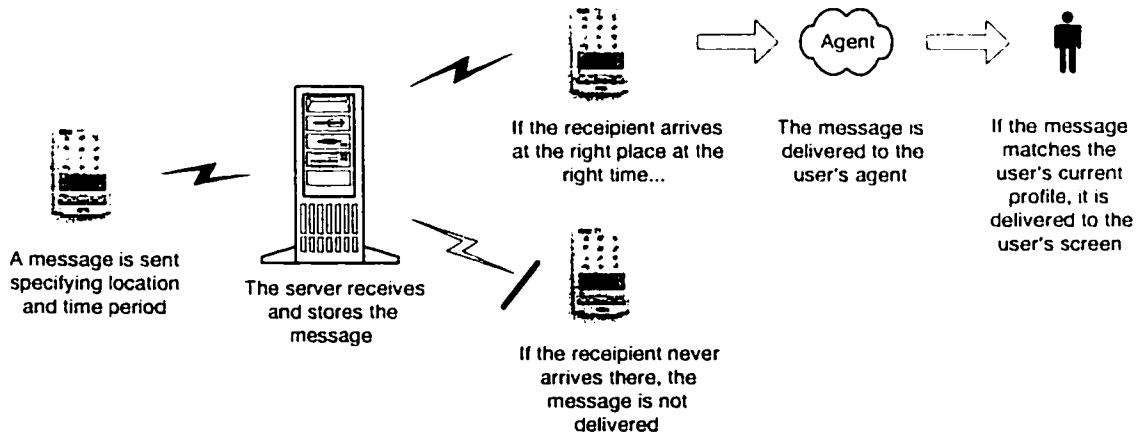


Figure 5.3: Primary Interactions Between the Clients and a Server

The primary interactions between client and server are illustrated in figure 5.3. The Hanging Message system assumes that there is a constant connection between the server and client devices, and that the server is continuously listening for commands from the

clients. Each client operates as both a sender and a receiver, relaying new messages to the server, and regularly querying the server for incoming messages. When the server receives a new message, it stores the message in a database, and it is indexed by location, recipient, and delivery time. Queries from clients specify these three attributes, and the server returns all matching messages to the client. The server then discards the query, so that it does not retain any knowledge of a client's location.

One of the interesting features of the Hanging Messages system is that it can send messages with a location-specific request to untargeted recipients. This allows messages to be sent that may be applicable to any users in a given location. One advantage of allowing untargeted messages is that companies can send promotions to any users who come to a certain location, without collecting individual user details. Another benefit of allowing unspecified recipients is that location based news or other information can be posted for the purpose of general interest such as traffic reports. From a client's perspective, for the purposes of message filtering, the information needs are categorized. Categories can be dynamically and interactively modified and they have different levels of specificity. A user profile on the Hanging Messages system consists of rules, which encapsulate user preferences. For example, a rule might state that a message should be passive or stored if it arrives before 9 A.M. In this case, the attribute is "time", the value is "before 9 A.M" and the consequent type is "passive". The attributes of a rule can be message category, time and location. Rules may address any number of message attributes.

The significance of the Hanging Messages system is that it attempts to use multiple context sensitive parameters such as location and time to deliver messages. This approach and the system show the possibility of developments an effective messaging service system according to the dynamics of the receiver's contexts. Message categorization and

its filtering for receiving the most appropriate message in a given context are the significance of the Hanging Messages system.

5.4 Other Related Research

"comMotion" [Marmasse, 01] is developed by MIT Media Lab's Speech Interface Group. It associates "to do" lists between a user's location and a virtual location, which are obtained from a GPS. Users can send each other reminders and query other user's location by means of a speech interface. "GeoNotes" [Persson, 01] allows users to mass-annotate physical locations with virtual 'notes', which are then pushed to or accessed by other users when they come into the vicinity of the location. This system provides an interesting model for message delivery in the form of push or pull. In a strictly push mode, users will keep the mobile device in their pocket, and when there is a high-ranking GeoNote in the vicinity, the device will signal and the GeoNote will pop up on the device. The user can change the push activity level to adjust the amount of messages. In a mixed push/pull mode, the GeoNote shows a simple diagram, which represents both what kind of and how many messages await in the user's location, on the screen and updates in real-time as the user moves. The user may click the device whenever he wants to check messages in his location. Lastly, in a strictly pull mode, the user actively searches for the GeoNote information.

5.5 Commercial Systems

Wherify[11] provides a location service that will enable customers to obtain real-time location information for individuals and property directly through the Internet or any phone. The small wristwatch sized equipment is specially developed and targeted for the people who need supervision: children, patients suffering from Alzheimer's disease, and memory loss personnel. It can also be used for animal tracking. Another interesting

[11] Wherify Wireless, Inc., http://www.wherifywireless.com/

application is InfoMove[12], which provides information services for drivers on top of other location services. With real-time traffic and incident notification, the user can have information in advance to avoid congestion and follow a re-routing. It also provides "turn-by-turn" driving instructions. All that the user has to do is to indicate the destination and then detailed real-time navigation information will be given to the user in spoken voice on "turn-by-turn" basis. Real-time vehicle monitoring and emergency assistance services are also used as part of the system. Saverfone[13] was developed in the United Kingdom and it allows users to find nearby promotions through WAP[14] phones. The shoppers can discover what special offers are going on at stores in their immediate area. The more enhanced application like MobileMate[15] from CTMotion has interesting features. M-Coupons, which is similar to the service from Saverfone, offers personalized electronic coupons and promotional messages to mobile users based on the user's profile, preferences and location. The Mobile Yellow Pages application allows mobile users to "pull" relevant location-based information via SMS[16] and/or WAP, to their mobile handsets at a time and place of their choice. After selecting the desired business, users receive contact information and directions to the selected destinations. In addition, the application has the Mobile Workforce Management Application, which is a unique management solution for those organizations with mobile field employees equipped with cellular handsets. This application enables managers and dispatchers to easily locate their mobile employees in real-time, using advanced on-screen map displays. Command and

---

[12] InfoMove Corporation, http://www.infomove.com/

[13] Brainstorm, http://www.saverfone.co.uk/

[14] WAP is the abbreviation for the Wireless Application Protocol that is an open, global specification, which empowers mobile users with wireless devices to easily access and interact with information and services instantly.

[15] CT Motion, http://www.ctmotion.com/

[16] SMS is the abbreviation for Short Message Service and is a service for sending messages of up to 160 characters (224 characters if using a 5-bit mode) to mobile phones. SMS is similar to *paging*. However, SMS messages do not require the mobile phone to be active and will be held for a number of days until the phone is active. SMS messages are transmitted within the same cell or to anyone with roaming service capability. They can also be sent to digital phones from a Web site equipped with PC Link or from one digital phone to another.

control information are communicated with individual or multiple employees, using the two-way SMS or Internet enabled messaging.

5.6 Summary

This chapter has introduced several related research efforts and commercial systems that deal with user's contexts. The approaches used in these efforts are parts of our context model. "Location-detecting-approaches" used in the Active Badge System, Locust Swarm and Hanging Messages can be used by the Context Agent in our system. The approaches in the Active Badge System and the Locust Swarm provide a way to detect a user's locations indoors. The approach in Hanging Messages, on the other hand, provides precise position information anywhere on the planet by using a Global Positioning System (GPS). Since our prototype system is not domain specific, one or more combinations of indoor and outdoor location detecting approaches can be used by a Context Agent according to the needs of the intended users.

Chapter 6

Concluding Remarks

In this last chapter we summarize the contributions of the thesis to making communications more effective and then briefly discuss future directions.

6.1 Contributions

The purpose of this research has been to define a model to capture contexts for a message delivery system and to show a system architecture that facilitates the creation of context- and rule-based communication software. The context models reported in the literature have been examined including a few commercial systems. The location context is used in the Active Badge System, which determines the indoor-location of an individual automatically by using a special tag and then forwards incoming calls based on that location information. The activity or the availability of a user is being used in the instant messengers controlled by a user to notify other connected users about the user's desirable status such as "busy", "be back", "away" and "lunch". The context based on time is used in certain electronic mail systems to choose the time to deliver e-mails. Although useful, these context mechanisms are limited in their aspect of context dynamics.

Major novelties of this thesis include a composite model of contexts for a message delivery system. This thesis describes the taxonomy of contexts for the purpose of message delivery, and then the notion of context is characterized. We built a prototype system as a proof-of-concept, which makes use of our notion of contexts. Detecting physical context and its interpretation for abstraction are simulated. Our prototype system is demonstrated and evaluated in a practical setup.

Among the contexts we have used, the locations are created in the form of a hierarchical tree to be used for creating user preference rules. A user, as a receiver of messages, explicitly states whether he is available or unavailable to receive a message at a given location. Based on preference rules exceptions are applicable. The time context being used in this thesis is not absolute time but task-based time, which is taken from the user's schedule. Moreover, the sociological context between a message sender and a receiver in the message delivery system is defined as a relationship hierarchy.

We have used the dynamic architectures based on the multi-agents and the tuple spaces, in which new software agents may be loaded and unloaded during execution. The system would have benefits from this architecture such as distributed processes and share-networked databases. Furthermore, the system can be enhanced without shutting down the system, which might be expensive in some cases.

The ultimate goal is to minimize the interruptions of a user by delivering the appropriate messages in a given context and holding or redirecting the other incoming messages. Our approach, which exploits personal traits and filter messages based on both a user's current context models and his/her preference rules, enhance the possibility of having a desirable delivery action for the user. It may reduce interruptions for dealing with incoming messages in the user's normal workflow. This aspect has to be evaluated, yet.

6.2 Future Research and Implementation

Several features, which would contribute to enhance the usability of the ACD system, were not implemented in this version due to time constraints. Since the main focus of the project was to establish a proof-of-the-concept, these features were deferred, to perhaps be implemented in the future.

6.2.1 Implementation with a Context Agent

A user's current contexts are simulated for this version of the system. To achieve the usability of this system, it should be implemented with a Context Agent, which actually detects a user's contexts. The ACD system can be combined with existing research or commercial systems as we discussed in the previous chapter. New approaches can be tested. A simple type of the Context Agent, which detects a computer's mouse movement, has been tested through our simulation program. The ACD client system allows a user to log into multiple machines in the network, then the system needs to know which machine the user is currently using. The Context Agent detects the computer's mouse movement and updates the user's location information in the TSpaces. So that an incoming call can be notified or forwarded to the user when necessary. It works well in a test environment, but we cannot assume that the user has total control for all of the logged-in machines in the real-world working environment.

6.2.2 Security and Privacy

Although using the TSpaces gives a great flexibility in the aspect of multiple agent system design, it reveals security weakness since it allows all the information to be shared. Some of the privacy-sensitive information such as the user profile should be protected. The TSpaces server provides access control by setting user and group permissions on the TSpaces. Therefore, users with the proper access control permissions may read and write tuples to and from the TSpaces. We may define and assign security

levels to the users (and/or the client machines), and protect the knowledge accordingly. In addition, the knowledge should be saved periodically into a secure repository for keeping knowledge consistency in case the TSpaces server fails.

6.2.3 Buddy-list Setting

Currently, a user has to input all the fields in the buddy list manually. It might be handy if an agent could help to add a buddy to the buddy list, and help to change it dynamically. A user interface agent for this purpose may suggest adding new buddies to the buddy list by monitoring the incoming and the outgoing calls (e.g. show the list of previous caller's phone numbers).

References

[Adams, 93] Norman Adams, Bill N. Schilit, Rich Gold, Michael Tso and Roy Want, "The PARCTAB Mobile Computing System", *Proceedings of the Fourth Workshop on Workstation Operating Systems (WWOS-IV)*, Napa, CA, October 1993, pp. 34-39

[Beigl, 00] Michael Beigl, "MemoClip: A Location based Remembrance Appliance", *Personal Technologies*, Vol. 4, No. 4, 2000, Springer Press, pp. 230-234

[Brown, 98] Peter Brown, "Triggering Information by Context", *Personal Technologies*, Vol. 2, No. 1, September 1998, Springer Press, pp. 1-9

[Chang, 01] Emily L. Chang, "Hanging Messages: Using Context-Enhanced Messages For Just-In-Time Communication", Master's Thesis, Dept. Computer Science, MIT, Massachusetts, 2001

[Chomicki, 00] Jan Chomicki, Jorge Lobo and Shamim Naqvi, "A Logic Programming Approach to Conflict Resolution in Policy Management", *Processing 7th International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufman, Breckenridge, Colorado, April 2000, pp.121-132

[CPL, 00] Jonathan Lennox, "CPL: A Language for User Control of Internet Telephony Services", Internet Draft, Internet Engineering Task Force, Nov. 2000, Work in progress

[Dey, 99] Anind K. Dey, Gregory D. Abowd, "Towards a Better Understanding of Context and Context-Awareness", GVU Technical Report GIT-GVU-00-18, *Graphics, Visualization & Usability Center, Georgia Institute of Technology*, 1999

[Graham, 01] Angus Graham, "Incremental Validation of Policy-Based Systems", Master's thesis, Dept. Computer Science, Concordia University, Montreal, 2001

[Kirsch, 97] Dana Kirsch and Starner T., "The Locust Swarm: An environmentally-powered, networkless location and messaging system", *Proceedings of the 1st International Symposium on Wearable Computers*, October 1997, pp. 169-170

[Lieberman, 01] Henry Lieberman and Ted Selker, "Out of Context: Computer Systems That Adapt To, and Learn From, Context", IBM Systems Journal, Vol. 39, No. 3&4, 2000, pp. 617-631

[Long, 95] Sue Long, Dietmar Aust, Gregory D. Abowd and Chris Atkeson, "Cyberguide: Prototyping Context-Aware Mobile Applications", ACM SIGCHI '96 Conference Companion, Vancouver, BC, Canada, April 1996, pp. 293-294

[Lupu, 97] Emil Lupu, Morris Sloman, "Conflict Analysis for Management Policies", *Proc. 5th IFIP/IEEE International Symposium on Integrated Network Management*, 1997, pp. 430--443

[Marmasse, 01] Natalia Marmasse, Chris Schmandt, "*comMotion*: a context-aware communication system", HUC 2000 Proceedings, 2000, pp.157-171

[McCarthy, 95] John McCarthy and Sasa Buvac, "Formalizing Context", Technical Report FS-95-02, The AAAI Press, 1995, pp. 119

[Miyata, 86] Yoshiro Miyata & Donald Norman, "Psychological Issues in Supporting Multiple Activities", *User Centered Design New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum Associates Publishers, 1986, pp. 266-284

[Pascoe, 97] Jason Pascoe, Johanna Moore, Ernest Edmonds, "The Stick-e Note Architecture: Extending the Interface Beyond the User", *International Conference on Intelligent User Interfaces*, 1997, pp. 261-264

[Persson, 01] Per Persson, Fredrik Espinoza and Elenor C., "GeoNotes: Social Enhancement of Physical Space", In Extended Abstracts (Design Expo), Conference on Human Factors in Computing Systems (CHI 2001), April, Seattle, 2001, pp. 43-45

[Schilit, 93] Bill N. Schilit, Marvin M. Theimer, Brent B. Welch, "Customizing Mobile Applications", *USENIX Symposium on Mobile and Location-independent Computing*, USENIX Association, Cambridge, MA, 1993, pp. 129-138

[Schmidt, 99] Albrecht Schmidt, Michael Beigl, and Hans-W. Gellersen, "There is more to Context than Location", *Computers & Graphics Journal*, Elsevier, Volume 23, No.6, December 1999, pp. 893-902.

[Want, 92] Roy Want, Andy Hopper, Veronica Falcao, Jonathan Gibbons, "The Active Badge Location System", *ACM Transactions on Information Systems 10(1)* 1992, pp. 91-102

[Ward, 97] Andy Ward, Alan Jones and Andy Hopper, "A new location technique for the active office", *IEEE Personal Communications 4(5)*, 1997, pp. 43-47

Appendix I : System Installation Guide

Common Requirements

1) The jdk1.3.1 must be installed
2) The TSpaces server (version 2.1.2) must be ready and connected

The ACD Client Requirements

-. The MiTAI Gateway server must be ready and connected

The ACD Server Configuration

1) Unzip **acds.zip** to C:\ACDS (it can be different, but you will have to modify paths in server.bat accordingly) on the server machine

2) Open **acds.bat** and ensure that all paths are correct (particularly the java path, as well as the class paths and the jar paths)

3) Specify a machine name of the TSpaces server at the end of the command if it runs on a different machine (the default name is the "**localhost**" that is the same machine as the ACD server)
ex. "**java** ... classpaths ... **acds.ServerMain** ***<TSpaces server name>***"

The ACD Client Configuration

1) Unzip **acdc.zip** to C:\ACDC (it can be different, but you will have to modify paths in client.bat accordingly) on a client machine

2) Open **acdc.bat** and ensure that all paths are correct (particularly the java path, as well as class paths and jar paths)

3) Specify a machine name of the TSpaces and the MiTAI Gateway server at the end of the command if it runs on a different machines (the default names are the "**localhost**" that is the same machine as the ACD client)

ex. "**java** ... classpaths ... **acdc.ClientMain** ***<TSpaces server name> <MiTAI Gateway server name>***"

Execution Procedures

1) Execute the MiTAI Gateway server
2) Execute the TSpaces server
3) Execute the ACD server
    3-1. Create a hierarchy of context
    3-2. Execute agents
4) Execute the ACD client
    4-1) Create and confirm user knowledge
        4-1-1) Relationship
        4-1-2) Buddy list
        4-1-3) Schedule
        4-1-4) User rule
    4-2) Set (simulate) current context, if necessary

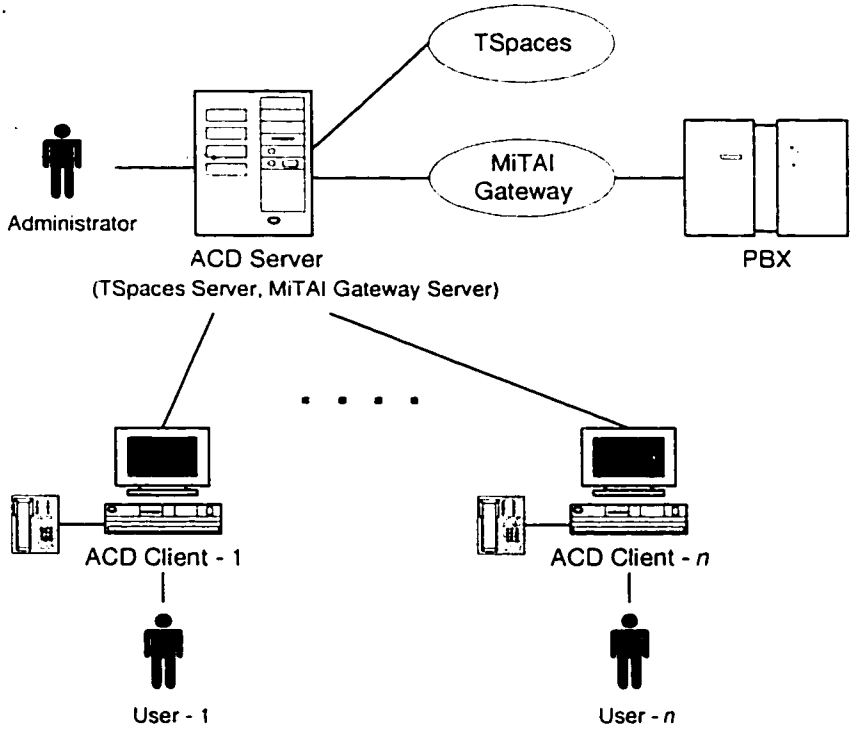


Figure – Example System Configuration

Appendix Ⅱ: Structure of Knowledge

Knowledge is stored in the TSpaces. The unit of knowledge is a "tuple", which is also the unit of data transaction with the TSpaces. A tuple is an aggregation of field(s), and maximum of seven fields are allowed by the TSpaces. The first field is reserved for the name of a tuple, and other fields can be either a simple string or a string with tags[17]. The simple string will be marked as 'String', and the string with tags will be marked as 'Tag String' in the definitions below. There are four tuples for this system: 'UserProfile', 'Call', 'Context', and 'AgentInfo'.

There are user-defined fields: the '*relationship*' in the 'UserProfile', the '*location*' and the '*activity*' in the 'Context'. The knowledge structure for those fields will be defined during run-time by using the GUIs. Those fields will be marked as the 'User Defined'. Some fields allow saving many same-format-fields. For example, the 'userrule' field in the 'UserProfile' tuple allows saving many user rules in that field. (marked as *)

1. UserProfile

1.1 Description
This is user specific information, includes basic user information, for call delivery.

1.2 Tuple fields

Tuple Name :: 'UserProfile'
+ id : String + userinfo : Tag String + relationship : Tag String – User Defined + buddylist* : Tag String + userrule* : Tag String + context : Tag String

- userinfo Field: This field contains fundamental user information for operating the system.

[17] Each field contains simple, but flexible data. It uses tags similar to XML format, but it does not use any particular parser due to the problems of TSpaces with XML. It uses internal function to have access to data instead of parsing.

Field Name :: 'userinfo'
+ passwd : String
+ fname : String
+ lname : String
+ office : String
+ home : String
+ cell : String
+ secretary : String
+ lunch time : String
+ meeting time : String

```
<?xml version="1.0"?>[18]
<!DOCTYPE userinfo [
  <!ELEMENT userinfo (passwd, fname, lname, office,
home?, cell?, secretary?, lunch time?, meeting
time?)>
  <!ELEMENT passwd  (#PCDATA)>
  <!ELEMENT fname   (#PCDATA)>
  <!ELEMENT lname   (#PCDATA)>
  <!ELEMENT office  (#PCDATA)>
  <!ELEMENT home    (#PCDATA)>
  <!ELEMENT cell    (#PCDATA)>
  <!ELEMENT secretary(#PCDATA)>
  <!ELEMENT lunch time (#PCDATA)>
  <!ELEMENT meeting time (#PCDATA)>
]>
```

- relationship Field: This field contains the hierarchy of personalized relationship information that is defined by a user using GUI.
- buddylist Field: This field contains a list of buddy and their information, which is set by a user

Field Name :: 'buddylist'
+ fname : String
+ lname : String
+ phone : String
+ relationship : String

```
<?xml version="1.0"?>
<!DOCTYPE buddylist [
 <!ELEMENT buddylist(fname,lname,phone,relationship)>
  <!ELEMENT fname   (#PCDATA)>
  <!ELEMENT lname   (#PCDATA)>
```

[18] XML DTD is for reference only. The implementation did not use these DTDs.

```
  <!ELEMENT phone    (#PCDATA)>
  <!ELEMENT relationship   (#PCDATA)>
]>
```

- userrule Field: This field contains a list of user rules, which are created by a user

Field Name :: 'userrule'
+ rulename : String
+ condition : String
+ fname : String
+ lname : String
+ phone : String
+ relationship : String
+ location : String
+ activity : String
+ time : String
+ action : String

```
<?xml version="1.0"?>
<!DOCTYPE userrule [
  <!ELEMENT userrule (rulename, condition, action)>
  <!ELEMENT rulename  (#PCDATA)>
  <!ELEMENT    condition(fname?,    lname?,    phone?,
relationship?, location?, activity?, time?)>
  <!ELEMENT fname(#PCDATA)>
  <!ELEMENT lname(#PCDATA)>
  <!ELEMENT phone(#PCDATA)>
  <!ELEMENT relationship(#PCDATA)>
  <!ELEMENT location (#PCDATA)>
  <!ELEMENT activity (#PCDATA)>
  <!ELEMENT time     (#PCDATA)>
  <!ELEMENT action (#PCDATA)>
]>
```

- context field: This field contains the user's current contexts, which are set by either simulation agent or context agent.

Field Name :: 'context'
+ location : String
+ activity : String

```
<?xml version="1.0"?>
<!DOCTYPE context [
  <!ELEMENT context (location, activity)>
  <!ELEMENT location(#PCDATA)>
  <!ELEMENT activity(#PCDATA)>
]>
```

2. Call

2.1 Description

This tuple is for call processing among agents.

2.2 Tuple fields

Tuple Name :: 'Call'
+ destagent : String
+ sourceagent : String
+ id : String
+ callinfo : Tag String
+ userrule* : Tag String
+ policyrule* : Tag String

- destagent Field: This field contains a name of the destination agent that will receive this tuple.
- sourceagent Field: This field contains a name of the source agent that is sending this tuple.
- id Field: This is a unique user id that using as a key value to scan and read tuples.
- callinfo Fieid: This field contains basic information of both a caller and a receiver such as phone number and name. It also contains relationship information between them.

Field Name :: 'callinfo'
+ callerinfo : String
+ phone : String
+ fname : String
+ lname : String
+ calleeinfo : String
+ phone : String
+ fname : String
+ lname : String
+ relationship : String

```
<?xml version="1.0"?>
<!DOCTYPE callinfo [
  <!ELEMENT    callinfo    (callerinfo,    calleeinfo,
relationship?)>
  <!ELEMENT callerinfo (phone, fname, lname)>
  <!ELEMENT phone (#PCDATA)>
  <!ELEMENT fname (#PCDATA)>
  <!ELEMENT lname (#PCDATA)>
  <!ELEMENT calleeinfo (phone, fname, lname)>
```

```
<!ELEMENT phone (#PCDATA)>
<!ELEMENT fname (#PCDATA)>
<!ELEMENT lname (#PCDATA)>
<!ELEMENT relationship(#PCDATA)>
]>
```

- userrule Field: This field contains user preference rules that are selected by agent(s). The format of this field is exactly the same as that 'UserProfile' has.

3. Context

3.1 Description

This is the hierarchy of context information that will be decided and input by system administrator. The contexts, which will be used in our prototype system, are a user's location, his/her current activity and the present time.

3.2 Tuple fields

Tuple Name :: 'Context'
+ location : String + activity : String

4. AgentInfo

4.1 Description

Agents are responsible for updating their status into the TSpaces. The SM agent uses this agent information when it decides the sequence of the process according to their availability.

4.2 Tuple fields

Tuple Name :: 'AgentInfo'
+ agentname : String + priority : String + status : String

```
<?xml version="1.0"?>
<!DOCTYPE agentinfo [
  <!ELEMENT agentinfo (agentname,priority,status)>
  <!ELEMENT agentname   (#PCDATA)>
  <!ELEMENT priority(#PCDATA)>
  <!ELEMENT status  (#PCDATA)>
]>
```