Developing Distributed Contextualized Communication Services

Edison Thomaz Junior

B.A., Computer Sciences

The University of Texas at Austin, 1999

Submitted to the Program in Media Arts and Sciences,

School of Architecture and Planning,

In partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

At the Massachusetts Institute of Technology

September 2002

Author

Edison Thomaz Junior

Program in Media Arts and Sciences

July 2, 2002

Certified by

Andrew B. Lippman

Senior Research Scientist

Thesis Supervisor

Accepted By

Andrew B. Lippman

Chairperson

Department Committee on Graduate Students

Developing Distributed Contextualized Communication Services

By

Edison Thomaz Junior

Submitted to the Program in Media Arts and Sciences,

School of Architecture and Planning, on July 2, 2002

In partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

Abstract

In the past few years, the worldwide adoption of digital devices such as computers, cell phones, media players and personal organizers skyrocketed. Due to advances in networking and computation technologies, we now have the opportunity to allow our devices to communicate and collaborate with each other in order to create an entirely new set of distributed user-centric services. An example of a distributed service would be a cell phone that learns more about social communication patterns by communicating with an email client application. This thesis demonstrates how we could develop such a system. I built a telephone application that benefits from the exchange of context information with a personal information manager to help users prioritize calls and make better-informed decisions about them. The application is based on a lower level specification that serves as the foundation for the design of sensible distributed services.

Thesis Supervisor: **Andrew B. Lippman**

Title: Senior Research Scientist, MIT Program in Media Arts and Sciences

Thesis Committee

Thesis Supervisor

Andrew B. Lippman
Senior Research Scientist
MIT Program in Media Arts and Sciences

Thesis Reader

Christopher Schmandt
Principal Research Scientist
MIT Media Laboratory

Thesis Reader

Ted Selker
Associate Professor
MIT Program in Media Arts and Sciences

Acknowledgements

I dedicate this thesis to **my family**, for their love, support and dedication.

Andy Lippman, my advisor, for giving me opportunity to come to the Media Lab and join this great research community. Andy has always pushed me hard to think outside the box and I really appreciate that.

Ted Selker, **Chris Schmandt** and **Henry Lieberman** for the thousands of hours spent discussing ideas with me and also for being part of my thesis committee, formally or informally.

Patrick Winston, for showing me what AI is all about, instigating my interest in the field and being such an example and role model in academia.

Greg Lavender, for being my undergraduate research advisor, for his inspiring lectures and for encouraging me to come to graduate school and apply to the MIT Media Lab.

Deborah Widener and **Polly Gugenheim**, for providing help and support in dealing with group administrative issues, as well as personal ones.

Linda Peterson and **Pat Solakoff**, for their patience and for answering all of my questions, offering good advice, telling me how to get things done in the lab and for genuinely caring about my well-being here.

Andrea Lockerd, for being my friend, my inspiration, for teaching me so much about so many things and for being deliciosa as only she can be.

Neil Murray, **Ingvar Aberg** and **Dah-Yoh Lim**, my friends at 8D, who came from nowhere exactly when I needed them the most. Thanks guys!

Surj Patel and **Jim McBride**, for being my good friends here in the lab, partners in crime and in lunch, technical consultants and for cheering me up when things didn’t look so bright. All at the same time and with their eyes closed!

Ernesto Arroyo, **Jorge Martinez**, the rest of **The Mexican Mafia**, **Kwan Lee**, **Shyam Krishnamoorty** and **Florian Mueller**, for joining the roller-coaster with me and keeping me sane and entertained in the Media Lab. It would be much tougher without you guys.

Peter Gorniak, **Stefan Marti**, **Natalia Marmasse** and the rest of the student committee gang. It has been a pleasure to work with them to make the lab a more enjoyable, fun and productive place to be.

My UROP **Dan Ports** for his enormous talent, contribution and patience in helping me get my thesis project off the ground. Good luck for you in your future adventures.

My good friends in Austin that I know I can always count on: **Katatau**, **Dani**, **Bilau**, **Juliana**, **Michel**, **Peu**, **Jabota**, **Elana**, **Diana**, **Lulu** and so many others. I love you all.

Table of Contents

1. INTRODUCTION 11

1.1. A HOLISTIC AND COLLABORATIVE APPROACH 12
1.2. CONTRIBUTIONS 13
1.3. CHALLENGES 15

2. BACKGROUND 17

2.1. ATTENTIVE SYSTEMS 17
2.2. CONTEXT-SENSITIVE ARCHITECTURES 19
2.3. TELEPHONY, MESSAGING AND AWARENESS 21

3. APPLICATION SCENARIOS 24

3.1. CALLER AVAILABILITY 24
3.2. EXPECTED CALL DURATION 25

4. SYSTEM ARCHITECTURE 27

4.1. INCALL 27
4.1.1. Implementation 27
4.1.1.1. Hardware 28
4.1.1.2. Software 29
4.1.1.3. inCall Telephony Core (iTC) 29
4.1.1.4. Full-Duplex Voice over IP 31
4.1.1.5. XML-RPC Client and Server 32
4.1.1.4. Ring Controller 33
4.1.2. Services 35
4.1.2.1. Availability of caller encoded in phone ring frequency 35
4.1.2.2. Expected duration of a call encoded in phone ring or frequency 39
4.2. SENSORAMA 44
4.2.1. Interfaces and Organization 45
4.2.1.1. High-Level Managers 46
4.2.1.1.1. Time Manager 46
4.2.1.1.2. Social Manager 47
4.2.1.1.3. Location Manager 49
4.2.1.2. Low-Level Managers and API 50
4.2.1.2.1. Activity Manager 50
4.2.1.3. Communication Protocol 50
4.2.2. Implementation 53
4.2.2.1. MacZoop 53
4.2.2.2. XML Parser 54
4.2.2.3. XML-RPC Objects 56
4.2.2.4. Sensorama Core and API 58
4.2.2.5. Interprocess Communication and Scripting (IPC) 58
4.2.2.6. XML-RPC Client 61
4.2.2.7. XML-RPC Server 61

5. EVALUATION 64

5.1. DESCRIPTION..64
5.2. USER STUDY GUIDELINES...65
5.2.1. Journalist 65
5.3.2. Travel Agency...67
5.3. RESULTS 69

6. CONTRIBUTIONS...71

7. FUTURE WORK..73

8. APPENDIX A: SENSORAMA API SPECIFICATION.....................................77

8.1. TIME MANAGER 77
8.2. SOCIAL MANAGER...77
8.3. LOCATION MANAGER 77

9. APPENDIX B: WORKLOAD DETECTION...78

10. APPENDIX C: EVALUATION FORMS..80

11. APPENDIX D: COMPILING LIBWWW AND XML-RPC-C..........................82

11.1. LIBWWW..82
11.2. XML-RPC-C...82

12. REFERENCES 83

List of Figures

Figure 1: inCall implementation architecture..27
Figure 2: inCall hardware..28
Figure 3: Internal inCall architecture 30
Figure 4: Call thread requests caller's time availability to caller's computer.................35
Figure 5: Call thread contacts the recipient's phone...36
Figure 6: Ring thread selects a ring tone 36
Figure 7: Recipient's phone sends context data to recipient's computer 37
Figure 8: Recipient's computer displays caller's context data......................................38
Figure 9: Recipient's phone agrees to start full-duplex..38
Figure 10: Full-Duplex starts on both ends...39
Figure 11: Call thread requests expected call duration to caller's computer..................40
Figure 12: Call thread contacts the recipient's phone..40
Figure 13: Ring thread selects a ring tone 41
Figure 14: Context data sent to recipient's computer 41
Figure 15: Recipient's computer displays caller's context data...................................42
Figure 16: Recipient's phone agrees to start Full-Duplex 43
Figure 17: Full-Duplex starts on both ends...43
Figure 18: Sensorama architecture 46
Figure 19: MacOS Sensorama implementation...53
Figure 20: User Study: Journalist Schedule...67
Figure 21: User Study - Travel Agent Schedule 68

List of Tables

Table 1: inCall Call Request Interface....33
Table 2: inCall Call Answered Interface....33
Table 3: Time Availability vs. Ring Tone....34
Table 4: Expected Call Duration vs. Ring Tone....34
Table 5: Time Manager API....47
Table 6: Email Number vs. Call Duration....48
Table 7: Social Manager API....48
Table 8: The Weather Channel URL....49
Table 9: Location Manager API....49
Table 10: Sample XML-RPC call....51
Table 11: Sample XML-RPC response....51
Table 12: Sample XML-RPC call with header....52
Table 13: Sample XML-RPC response with header....52
Table 14: Using the Expat parser....55
Table 15: Sample usage of XML-RPC Objects....56
Table 16: Output of sample usage of XML-RPC Objects....57
Table 17: List of XML-RPC Objects....57
Table 18: Sample Applescript....59
Table 19: Using the XML-RPC Client....61
Table 20: XML-RPC Server dispatch calls....62
Table 21: inCall vs. IP Phone Comparison for the Pairs....70
Table 22: Time Manager API....77
Table 23: Social Manager API....77
Table 24: Location Manager API....77
Table 25: Metric to determine workload....79
Table 26: Determining a workload....79

1. Introduction

In the last decade, the worldwide adoption of digital devices such as computers, cell phones and personal organizers skyrocketed. Where just a few years ago possessing a cell phone or handheld device could easily award you special treatment in restaurants and social circles, today virtually anyone can walk into an electronics store with less than $300 dollars and leave with a variety of digital gizmos. The numbers confirm this trend. According to a September 2001 J.D. Power and Associates study, 52 percent of households in the 25 largest urban markets surveyed in the United States now have at least one cell phone. In some European countries such as Iceland and Finland, the figure exceeds 75 percent. Consumers bought an estimated 6.4 million digital cameras and 12 million digital organizers in 2001 [1].

The pervasiveness of small computational devices does a lot more than simply add weight to our pockets. It represents a new and important trend in the consumer and computational spaces. As the complexity and number of digital devices that we carry with us increases, so does the amount of distributed computation, media and information that continuously surrounds us. Consequently, as time goes by, this digital "aura" can become more connected to our lives and meaningful to us. As an example, MP3 devices, digital cameras and organizers are powerful computing systems that carry some type of information that represents our values and way of life. The collection of songs in our MP3 player expresses our musical interests and therefore our personality to some degree. Digital cameras have the means to infer where we have been physically lately and the people we share our lives and experiences with. Finally, our personal organizers know what our upcoming appointments are, who we usually communicate with on a regular basis and much more.

We could think of these digital devices simply as service providers, but the tight coupling that exists between them and our lives turns them into implicit pointers to our personality, our activities, our physical health and even our state of mind. These devices and the bits that they carry represent a largely ignored resource that could be translated into new kinds of useful device functionalities.

1.1. A Holistic and Collaborative Approach

Although opportunities exist, realizing the full potential of these digital devices and getting them to represent our selves and different aspects of our lives is not a simple task. They need to be designed to, for example, not only play MP3s but to also make some kind of sense of what they are playing and under what context. Fortunately, an added bonus of the computational surplus in our personal digital devices is that in addition to providing a fixed set of services such as playing music or making phone calls, devices could perform other peripheral tasks without any impact on performance. Collecting observational data about where and how they are used, analyzing what kinds of data they manipulate and also interacting with other devices in the surrounding area are examples of such tasks. The ability to collect and analyze usage data is particularly significant because digital devices could then make educated guesses about the state of users and their environment, turning themselves into continuous sensors with a bit more of awareness of their purpose in our lives and a better understanding of the media and information that they host. Digital devices with these characteristics are commonly referred to as context-aware, or attentive [15], but I like to call these systems "sensible". I believe that in the foreseeable future, most of our digital devices will be sensible to some degree. Incidentally, I also think that the definition of what we today refer to as analog and digital will be determined in the years to come much more in terms of the ability of devices to be sensible and aware of their environment than by the materials and techniques employed in their physical construction.

If we think of all the devices that we use on a daily basis as a collection of sensible systems, in additional to the enormous amount of implicit user information that can be collected and used by each single sensible device, we could also combine the observations and inferences of independent devices to design a whole new class of collection-level services. These services would rely on the sharing of information and collaboration between two or more devices to be realized. For instance, it could be useful to let our smart credit card inform our heart beat rate monitor that we had a greasy hamburger and milkshake for lunch. We might have to work twice as hard in the gym to get rid of those extra calories. Likewise, it could be useful for our telephone to communicate with our personal organizer and prioritize our phone calls according to the activity of our contact manager and email client application. Although the opportunity clearly exists for cross-device communication and information sharing at a cognitive level, very few devices today could be considered sensible. In other words, very few devices actually *collect, analyze and share* usage data.

1.2. Contributions

This thesis demonstrates the design, implementation and evaluation of inCall, a telephone application that benefits from the exchange of context information with a personal information manager to help users prioritize incoming calls and make better-informed decisions about them. At a higher level, InCall can be divided in two main components: *iTC* and *Sensorama*. ITC is the core unit of inCall, responsible for the voice over IP [47] service, logic and the overall functionality of the telephone system. Sensorama is an application and interface that allows a computer personal information manager to become sensible and exchange user context information with other devices, such as a telephone.

The key idea behind InCall is that when user A places a call to user B, inCall communicates with user A’s computer and collects observational context information about user A, such as *how* busy he or she is, *where* he or she is and *who* he or she communicates with on a regular basis. This information is then sent to user B so that user B can better prepare for the call or make a better informed decision about whether to take the call or not.

Features of inCall include, considering a call from user A to user B as above:

- Communication with the personal organizer of A and encoding of time availability of A into ring type when the phone of user B rings indicating an incoming call. The goal is to eliminate long phone-tags between users of the system who want to talk to each other.

- Communication with the email client application of A and encoding of social communication patterns of A into ring type when the phone of user B rings indicating an incoming call. The system changes its ring type according to how long a phone call is expected to last. This deduction is based on how often individuals communicate with each other by email. The goal of this feature is time prioritization.

- Communication with information sources to determine the weather conditions of where A is located based on the zip code of A. This piece of context information can be displayed to user B. The goal of this feature is to give user B an idea of the physical context of the call.

- Display of context-information about A to B when the phone of B rings. In addition to changing its ring type based on context information, the system also displays the context cues mentioned in the features above in a screen before calls are answered.

One key aspect of inCall is that *it enables user context information to flow from the caller to the callee*, unlike most personal awareness and phone applications built so far. This is an important distinction between inCall and other telephony and messaging applications with user-awareness functionality built-in.

Moreover, inCall shows how a telephone network employing a voice over IP architecture can be used to easily provide user services that require not only the transmission of voice but also of data. In the case of inCall, voice packets are sent interchangeably with data packets in order to frame a context for a call that is ongoing or that may take place in the near future.

1.3. Challenges

Developing a system such as inCall is challenging for several reasons. One of them is that there is not a standardized communication platform that is designed specifically for the purpose of developing collaborative and distributed services that depend on a group of devices or software applications. New emerging hardware technologies such as BlueTooth [37] and Wi-Fi, more commonly known as 802.11 [38], are now able to provide the physical link that enables devices to exchange data with each other. Distributed protocols and language specifications for remote method invocation such as XML [39], XML-RPC [40] and SOAP [41] could be seen as a common denominator that enables software applications to execute programs in a distributed fashion. Ideally, a combination of these different technologies is what would be considered an ideal architecture for distributed services such as inCall. Incidentally, inCall uses XML-RPC as its remote method invocation and messaging protocol.

Another challenge has to do with the acquisition of context information. Collecting contextual user information from devices and software applications is not a simple task. The number of systems that implement an interface to user-centric data such as patterns of usage is very small. Most of the systems we use are black boxes, running proprietary software, which were not designed to give any visibility of its inner workings. That is the main reason why Sensorama had to be developed. Sensorama, described in detail in a later section, not only collects context information from Microsoft Entourage for the Macintosh [48], a personal information manager, but also analyzes this information and makes it available to other devices that implement the XML-RPC protocol.

One of the primary challenges of developing a system like inCall is that its efficiency and effectiveness can be hard to evaluate over a short period of time. This is due to the fact that the evaluation process in these cases needs to be qualitative for the most part. In the evaluation section of the thesis, I describe how I attempted to study inCall quantitatively by means of a series of 30-minute user study.

One of the fundamental ideas that I promote with this thesis is that an architecture that permits digital devices to talk to each other is highly desirable. However, a new trend in the personal digital device space could weaken the role of personal digital devices as independent systems. Due to the variety of mobile devices that many people are starting to carry with them on a regular basis, such as cell phones, PDAs, MP3 players and still cameras, consumer electronic companies have begun to investigate whether it makes sense to combine several of these distinct devices into one. An example of a device that unites a cell phone with a personal organizer in the same form factor is the Handspring Treo [44]. However, integrating several devices into one poses a considerable challenge to technology companies and designers. While the miniaturization of electronic circuits continues, it is increasingly harder to create user interfaces for devices that are smaller and feature-richer than previous ones. Therefore, it may well be the case that personal digital devices will remain application-specific for the foreseeable future.

Personal privacy cannot be ignored as another very significant challenge to inCall and similar context-based applications. Exchanging sensitive information among a group of devices poses a series of security and authentication questions with regard to the robustness of distributed systems. Users should be completely aware of what kinds of contextual information about them are being used, by whom and at what times. These issues cannot be disregarded by any means. However, due to the nature of this thesis, it is primarily focused on showing how context information could benefit communication and distributed services. Exploring how to address these security and privacy issues in a distributed environment would be a perfect way to bring inCall closer to fruition and to end users.

2. Background

2.1. Attentive Systems

There is an extensive body of knowledge in the area of context awareness, among them COACH, The Lumiere Project and Letizia. All these systems are significant because their key functionality stems from the utilization of context-based information.

The Cognitive Adaptive Computer Help (COACH) [17] is an example of an attentive system. It is an adaptive help system that monitors a user's actions. When a user begins an unfamiliar task, COACH will proactively present advice to the user. Users may also explicitly request help information. In either case, COACH uses a user model to estimate the user's level of experience with the current task, and then chooses articles from a database of help information. COACH is significant because it was one of the first context-aware systems developed.

In the last few years, a wide variety of agent-based systems were developed, with the intention of offloading common tasks to autonomous intelligent software applications, such as personalized news retrieval and filtering. Their agents relied heavily on observations and user context information to execute their semi-autonomous jobs; as a result, they are also attentive systems. With agents, instead of user-initiated interaction via commands and direct manipulation, the user is engaged in a cooperative process in which humans and computer agents both initiate communication, monitor events and perform tasks. Pattie Maes was one of many research pioneers who developed a variety of software agents in order to explore and study the human computer interface paradigm changes that are intrinsically connected to the agent computing model [14]. In particular, she was interested in issues such as competence and trust that arise when humans and semi-autonomous agents work together. Maes was one of the first to try to understand the nuances and subtleties in developing agents that interact with humans on a day-to-day basis.

Most agent systems are behavior-controlled by user actions and activities, especially user interface agents; therefore they should also be considered attentive systems. The Lumiere Project, developed by Eric Horvitz at Microsoft Research, centers on harnessing probability and utility to provide assistance to computer software users [11]. It uses Bayesian user models to infer a user's needs by considering a user's background, actions, and queries. Several problems were tackled in Lumiere research that are very relevant to the Sensorama and inCall, including the construction of Bayesian models for reasoning about the time-varying goals of computer users from their observed actions and queries, gaining access to a stream of events from software applications, developing a language for transforming system events into observational variables represented in Bayesian user models, developing persistent profiles to capture changes in a user's expertise, and the development of an overall architecture for an intelligent user interface. Lumiere prototypes served as the basis for the *Office Assistant* in the Microsoft Office '97 suite of productivity applications.

Letizia, developed at the MIT Media Lab by Henry Lieberman, is an observational software agent that compiles a history of how users utilize their web browsers [13]. Then, based on the history of previous web pages visited, the agent tries to infer which links in a new page the user will most likely want to visit. With that empirical information, the agent pre-caches the web pages that it thinks the user will want to see. Expert Finder, also by Lieberman, is another agent that observes users as they interact with their software applications and then automatically classifies both novice and expert knowledge by autonomously analyzing documents created in the course of routine work [10]. Letizia and Expert Finders are important research projects because they demonstrate in practice the degree to which agents can be used to monitor users engaged in human-computer interface activities in order to assist them and help them be more productive.

Observable APIs are defined as interfaces that permit a system to watch and be notified about changes in another system. Most software systems today do not offer any kind of observability to its external world. As an example of what I mean by software observability, there is not any generic way to programmatically query a media player and ask it which songs a user has been listening to lately. This is unfortunate, since software applications usually provide very valuable contextual information about users. All this contextual information is lost unless there is a way to have access to it. Moreover, as we move forward and attempt to create systems that profile users and predict their actions, it is absolutely essential that we monitor them dosely, at the application usability level. Cameo is a C++ toolkit and associated model that allows programmers to build observable APIs [23]. The development of the Cameo environment has been largely influenced by a technique called status/event analysis [24], where systems can be decomposed into parts that have a set of status and events associated with them. Sensorama was influenced by Cameo, especially its *Manipulation* primitive. Due to its XML-RPC interface, Sensorama offers only a procedural query interface, as opposed to a callback registration interface. Also, Sensorama is much more specific than Cameo, because it was built to realize some real user scenarios in a distributed environment.

2.2. Context-Sensitive Architectures

The context-sensitive architectures mentioned here are particularly relevant to the Sensorama component of the inCall system. These architectures aim to facilitate the development of context-aware applications. Sensorama could be generalized to be presented as an architecture similar to the ones below rather than as a support application and interface for inCall.

The MIT Laboratory for Computer Sciences Oxygen Project is a major ubiquitous computing research initiative whose goal is to re-architect software and hardware systems to make them much more user-centric than they are today [27]. The proposed distributed architecture is based on a collection of devices that work together seamlessly. A major priority of Oxygen is to develop an infrastructure where people offload tasks to the environment around them. The Oxygen Project encompasses research all the way from chip design to human interfaces and computer networks. Here is a complete description of the project:

"Oxygen is an integrated collection of eight new technologies: handhelds, wall and trunk computers, a novel net, built-in speech understanding, knowledge access, collaboration, automation and customization. The power of Oxygen lies not in any one piece but in the totality of these human-oriented technologies together. They forge a new computing metaphor that we hope will mark an important shift from the desktop and icons of today, as those innovations did from text-only systems"

The Context-Toolkit was developed to make it easier for developers to create context-enabled applications [28]. According to Salber, Dey and Abowd, context is defined as the information that is part of an application's operating system and that can be sensed by the application. What is novel about the Context-Toolkit is that it was designed to model graphical user interface (GUI) toolkits. For instance, the same way that GUI toolkits are constituted of widgets that provide an application interface to users interactions, the Context-Toolkit also takes advantage of the widget design pattern to insulate applications from context-sensing mechanisms. The Context-Toolkit focuses on a new way to design and build new sensors, based on widgets. These sensors can then be integrated into applications.

CyberDesk is a desktop platform on top of which context-aware software applications can be built [29]. It is aimed at providing a more flexible framework for integrating software behavior. Software applications register their services in the *Registry* and CyberDesk becomes in charge of signaling to the user which services are available relative to the task being performed. The advantage of developing CyberDesk applications has to do with the user interface. Rather than displaying all the available services to the user at all times, the interface is limited to displaying those services that are relevant to the user's current context.

2.3. Telephony, Messaging and Awareness

The systems described below are telephone and awareness applications that either take advantage of context information in the telephony domain like inCall or show the benefits and flexibility of voice of IP.

Over the years, a wide range of media space systems has been created [8]. The Awareness Community Portal is one example of a shared media space system, which displays information in shared physical spaces and attempt to provide a feeling of social awareness [16]. The Portholes project is another media space system, which investigated how media space technologies support awareness through the exchange of images [5][9]. These awareness systems are relevant because like inCall, they attempt to convey a sense of awareness by transmitting user context information from point-to-point.

A variety of telephone and messaging applications have attempted to benefit from context information. Quiet Calls offers a new form of communication, extending the choices offered by synchronous phone calling and voice mail [18]. It is a technology that allows mobile telephone users to respond to telephone conversations without talking aloud. The system developed by Nelson, Bly and Sokoler addresses the typical problem of having to establish a phone communication with someone in environments that are not suitable for conversations or talking aloud, such as conference rooms and meetings. InCall and Quiet Calls are similar for two reasons. Firstly, they are both illustrative of how our communication needs and settings are changing rapidly and what we are doing to deal with these behavior changes. Secondly, both systems are context-dependent; inCall tries to deduce the context of users while Quiet Calls receives direct input from users about their context and availability.

The Live Addressbook is an application that helps users make more informed telephone calls by providing callers with presence cues of the people that they want to call [4]. This allows participants to negotiate a call before it actually takes place. The application accomplishes this by using the “buddy list” concept from instant messaging applications with the addition of user-controllable presence indicators. The key difference between the Live Addressbook and inCall is that in the case of inCall, contextual information about the caller is sent to the callee whereas in the Live Addressbook, contextual information about the callee is sent to the caller. Incidentally, this reversibility in user context flow is a key aspect of this research.

ConNexus goes a bit further than the Live Addressbook above and presents specific information about callee schedules and activities, such as for how long the callee has been idle or whether the callee has upcoming appointments [20]. Therefore, a lot of information is available to the caller before making a phone call. The caller also knows beforehand what he or she can expect when placing a phone call. Again, just like the Live Addressbook, The key difference between ConNexus and inCall is that in the case of inCall, contextual information about the caller is sent to the callee whereas in ConNexus, contextual information about the callee is sent to the caller.

As far as message filtering is concerned, CLUES observes user communication patterns and other user activities to prioritize and present messages in order of importance and timeliness [19]. CLUES helps to identify important messages based on time-varying information sources. The biggest challenge of CLUES is to learn user's short-term interests and prioritize incoming messages accordingly. In order to determine user's interests, CLUES relies on information found in a user's work environment such as calendars, email and call logs and rolodexes that we could comfortably refer to as context information. CLUES is not exactly a phone application, but the type of context information used by CLUES is of the exact same kind that inCall relies on to make inferences about phone callers.

The ICEBERG project of the University of California at Berkeley is looking into issues associated with the converged network of the near future, specifically the migration of telecommunications networks towards Internet technology and voice over IP [26]. On one hand, one of the goals of ICEBERG is to serve as a foundation for applications integrating data and voice. On the other hand, ICEBERG attempts to support diverse access technologies (such as the Public Switched Telephone Network, digital cellular networks, pager networks, and IP-based networks). InCall and Sensorama together explore the potential of voice over IP services, but neither has the low-level computing and telecommunication infrastructure focus as ICEBERG.

IMPROMPTU is an IP-based audio platform for mobile communications and networked audio applications developed at the MIT Media Lab by Kwan Lee [12]. Several similarities between inCall/Sensorama and IMPROMPTU exist. Both projects were designed with scalability in mind and around an architecture that serves as the foundation for distributed services. Moreover, the Full-Duplex component of inCall stems from the IMPROMPTU chat application, with a few minor modifications. However, the motivation behind the two projects is significantly different. While Sensorama was conceived as an integration infrastructure for personal digital devices, with inCall as an example of the benefits that such as infrastructure can provide, Kwan Lee was primarily interested in exploring how to develop audio interfaces for a sophisticated mobile platform with IMPROMPTU.

3. Application Scenarios

InCall is a phone systems distinguished by its set of context-sensitive user services. The usage scenarios below portray how inCall might be used and how people in real life situations could benefit from it. The fictitious characters in the application scenarios below are James, Bridget and Lucy. James and Bridget are married and have one daughter, 12-year old Lucy. James runs a small consulting business with 5 others and Bridget works as a dentist at the local university. They work in opposite sides of town, so they get to see each other only in the morning for breakfast and sometimes late in the evening. Therefore, they rely on their office and cell phones to communicate with each other during the day. It is common for Bridget to call James several times a day, during her breaks and when patients are late for appointments. Conversations between the two are more often than not about their personal lives, such as who is going to take the kids to school the next day and whether they are going to visit grandma during the holidays. Most of the time, however, their conversations are fairly short; they don't talk for more than 5 minutes at a time. After all, James is usually very busy at work, running to and from meetings and conference calls all day long. Lucy also calls her parents regularly during the day. Luckily for James, who does not have an administrative assistant, his company has an inCall voice communication system installed, which facilitates the arduous job of prioritizing between so many different tasks that need his attention and time throughout his day. InCall replaces the old phone system with a smart and context-sensitive communic ation device.

3.1. Caller Availability

It's lunchtime. Bridget picks up her cell phone and punches her husband's office number. For years during her lunch break, Bridget has been calling her husband to see how his day is going and talk about the kids and what they are all doing for dinner. Instead of hearing James' voice however, she is sent to voice mail. Most of the time, James spends his lunch break in the city park, enjoying the scenery, the cool breeze and his turkey sandwich. Not today. He is busy in an important lunch conference call with a group of important partners and cannot talk to his wife. Thanks to inCall, though, when his wife's call comes through, the type of ring tells James how busy Bridget is and whether would be ok to call her back later. He glances at his phone caller ID and confirms that Bridget is the one on the line. His phone emits the sound of someone snoring, which means that she will not have any appointments for the next 3 hours. It is a slow day in the office for her. That is perfect, because he will be free later in the day. At 4PM, James calls Bridget and they decide to go have pizza for dinner with the kids.

inCall interfaces and communicates with the personal organizer of its users, allowing it to encode time availability into ring type, eliminating long phone-tags between people who want to talk to each other.

3.2. Expected Call Duration

Every week, James needs to fly to California to visit some of his clients located there. James is a first class frequent flyer. Because of how often he travels, James devised a special routine for his traveling days that allows him to do some work before heading to the airport. He usually goes to the office around 8AM, works for a couple of hours and then leaves to catch the 12PM flight to San Francisco. One of the problems that James had before using inCall was that sometimes he was so busy in the office that he had to force himself to stop working or talking on the phone so as not to miss his plane. This was particularly true when he received an important phone call minutes before leaving to the airport and could not hang up the phone right away. As with many, being late is an extremely stressful condition for James. After the installation of inCall, however, his job of prioritizing communication tasks became a lot easier, especially under time-critical circumstances. Now, when he is about to leave the office for an upcoming appointment or flight and has only a few minutes to spare, James avoids answering the phone when its ring type indicates a long phone call.

inCall can be programmed to change its ring type according to how long a phone call is expected to last. Because of its ability to communicate with email client applications, inCall is aware of whom one talks to on a regular basis and whom one might have to talk to for a longer period of time.

4. System Architecture

InCall is a service that derives its unique value from the communication and sharing of data between personal digital devices, in this case a personal computer and a telephone. InCall can be divided in two main components: *iTC* and *Sensorama*. ITC is the core unit of inCall, responsible for the voice over IP service, logic and the overall functionality of the telephone system. Sensorama is an application and interface that allows a personal information manager to become sensible and exchange user context information with other devices, such as a telephone. Sensorama is the source of context information for the telephone system of inCall. In this chapter, we first describe inCall, then Sensorama.

4.1. inCall

4.1.1. Implementation

The inCall system was implemented as a mobile platform and also as a desktop system. The mobile platform was created around the iPaq personal digital assistant and the desktop one ran in a variety of Red Hat 7.2 Linux boxes in the laboratory.

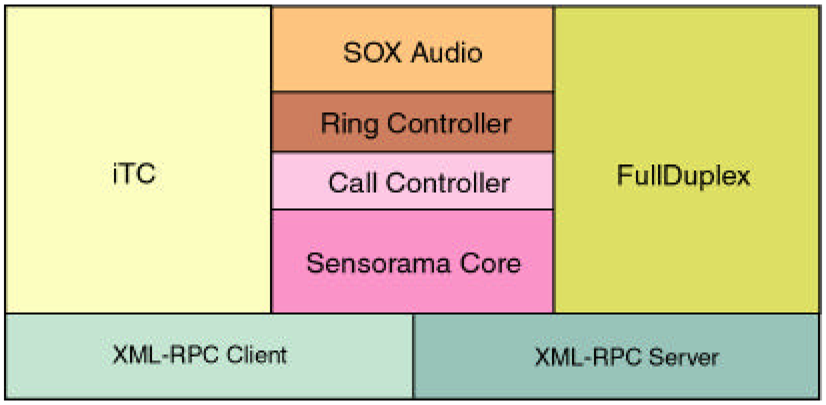


Figure 1: inCall implementation architecture

4.1.1.1. Hardware

The iPaq used to implement the system was the COMPAQ iPAQ H3600. A dual-slot PC Card expansion pack was added to the iPaq and the two extra PC slots were filled with a LUCENT Wavelan 802.11b wireless card and a 1GB IBM Microdrive. On the desktop side, standard PC boxes with network connectivity were used. Below is a picture of the iPAQ, the 802.11b card and the Microdrive.

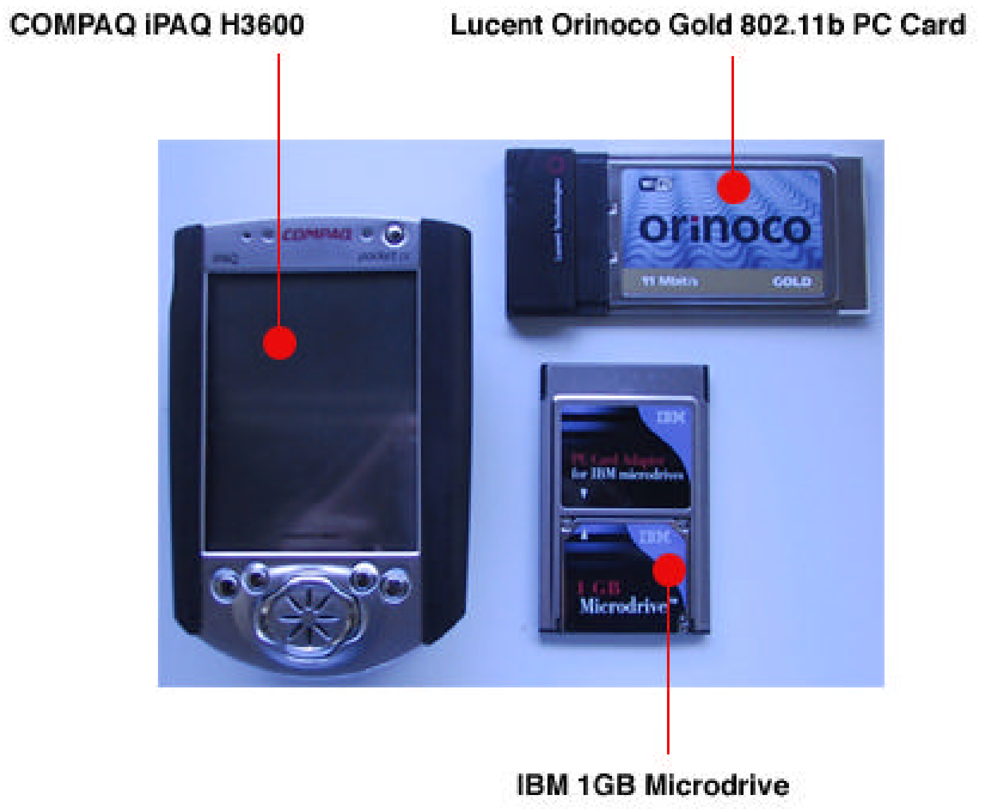


Figure 2: inCall hardware

4.1.1.2. Software

The desktop version of inCall was developed under the Red Hat Linux 7.2 operating system, but it should run without problems under other Linux distributions as well. The mobile inCall system was implemented on top of the Familiar v0.5.2 Linux distribution [31]. The iPaq is a device with little internal memory and storage space compared to today's desktop and laptop computers. This makes it very hard to install in the iPaq all the necessary tools and libraries required to develop relatively large software systems. To alleviate this situation and support software development for the iPaq running Linux, Compaq Research set up a group of external compilation servers called skiffclusters. Skiffclusters are supercharged iPaqs, with a lot of memory and storage space that give developers an unconstrained environment that allows them to create software without too many restrictions. Once a piece of software has been compiled in the skiffclusters with all the resources that it needs, it can then be transferred to the local iPaq using FTP. InCall was developed using these external skiffclusters in C and C++.

4.1.1.3. inCall Telephony Core (iTC)

The inCall Telephony Core, also called iTC, is the heart of inCall. Among the responsibilities of the iTC are handling the connection negotiation, managing the exchange of context data, using logic to select and trigger the appropriate ring tone and establishing the full-duplex audio connection.

It provides two interfaces for control: a command-line shell prompt and an XML-RPC server. The command-line shell is used when placing or answering calls, while the XML-RPC server interface is used for communication with other devices and for the exchange of context data relative to originating or incoming calls. Each of the interfaces runs in its own thread and waits for input. The input then triggers the appropriate command handlers, which often involves starting up new threads. These two control threads run continuously whenever the system is running.

There are three threads that can be invoked inside the command handlers: the calling thread, the ringing thread, and the full duplex thread. No more than one of those threads can be running at any one time.

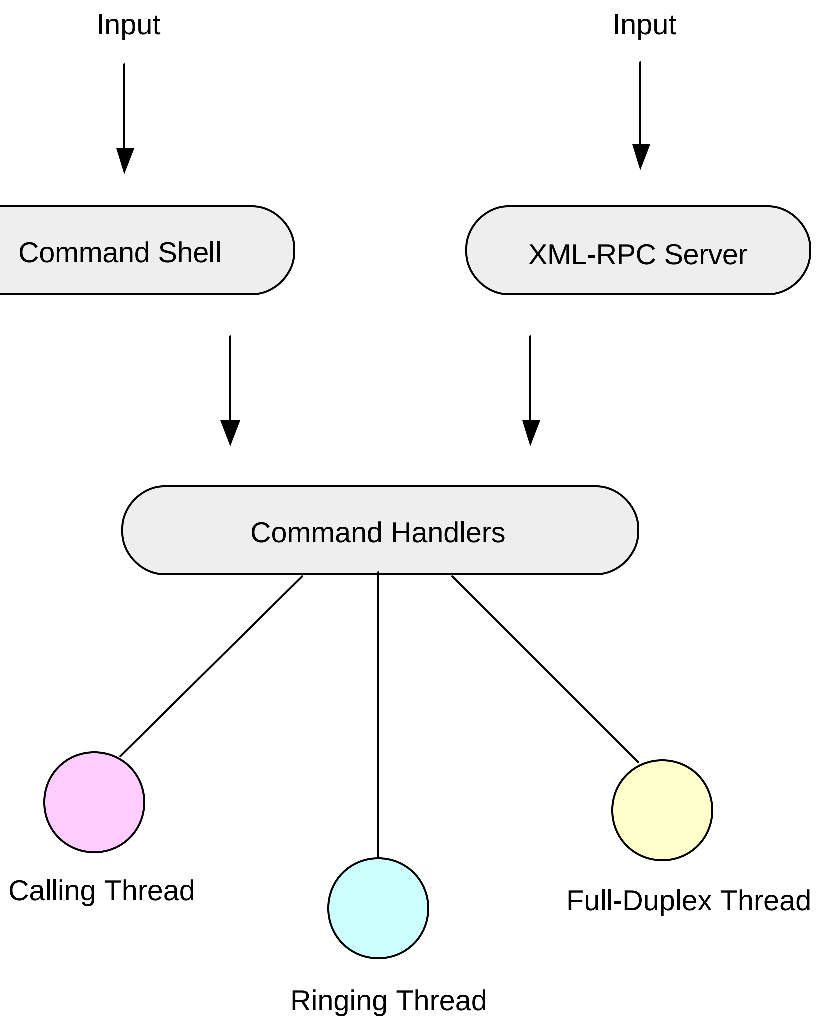


Figure 3: Internal inCall architecture

The system can be in one of five states: *idle*, *calling*, *ringing*, *connected*, or *transition*; each of these are pretty much associated with one of the three threads started by the handlers.

Each thread locks a mutex when it changes the state, and unlocks it when it shuts down. Most commands can only be used in the idle state. For example, one cannot make a call when the phone is already in use. The transition state is used to prevent race conditions when switching from one thread to the other; the new thread hasn't started up yet, but the system isn't idle either.

4.1.1.4. Full-Duplex Voice over IP

The voice over IP (VoIP) component of inCall is based on previous work done by Kwan Lee [12] in his Master's Thesis titled "IMPROMPTU, Audio Applications for Mobile IP". IMPROMPTU, also based on the iPaq, is defined by Kwan as a platform for mobile communications and networked audio applications. One of the applications developed for IMPROMPTU is a full-duplex telephone system. This is the application inCall stems from. Internally, the full duplex phone application is actually very simple. When a connection is established, two completely independent threads are started, a sending thread and a receiving thread. The sending thread is responsible for constantly capturing digitized audio from the iPaq built-in microphone, creating UDP packets with the audio data and sending these packets to the other end of the connection [30]. The receiving thread, as expected, receives incoming UDP packets, extracts the audio data from them and places the data in the iPaq audio buffer. These two threads permit the simultaneous sending and receiving of voice in real-time, effectively morphing the iPaq into a telephone. Because of network congestion or other potential problems, the Full-Duplex threads were designed to time-out if no data is received within 5 seconds.

4.1.1.5. XML-RPC Client and Server

What differentiates inCall from current phone systems is that it is able to modify its behavior depending on its own state and the state of its users. In order to communicate with Sensorama or with other devices and exchange contextual information with them, inCall was designed from the ground up with an embedded XML-RPC client and server.

The inCall XML-RPC client and server are based on an XML-RPC library for C and C++ referred to as xml-rpc-c [32]. This library defines itself as a lightweight RPC library based on XML and HTTP. The client side of xml-rpc-c requires the installation of libwww. Libwww is a highly modular, general-purpose client side web API written in C that is maintained by the W3C consortium [33]. The server side of xml-rpc-c requires the installation of ABYSS, a small footprint, fully HTTP/1.1 compliant web server [34]. Xml-rpc-c installs the ABYSS server by default.

The inCall XML-RPC client is used to communicate with other devices as part of the handshake and negotiation protocol that takes place when users implicitly or explicitly utilize inCall services. An implicit utilization of an inCall service would be answering a phone call, while an explicit utilization would be making a phone call, for instance.

With regard to the inCall XML-RPC server, it runs in its own thread and spawns new threads in order to serve requests. Not unlike the client, the server is also critical to communication and negotiation when inCall services are in progress, about to be in progress or terminating. The inCall XML-RPC is bound to the following interface:

Boolean CallRequest (Args)
Where **Args**:
String pIP, String dIP, Integer htnEvent, Integer mtnEvent, Integer eDuration

Table 1: inCall CallRequest Interface

And

Boolean Call Answered (String pIP, String dIP)

Table 2: inCall CallAnswered Interface

The arguments of CallRequest are pIP, dIP, htnEvent, mtnEvent and eDuration, for the IP address of the caller's phone, the IP address of the caller's personal computer, the number of hours until the caller's next appointment, the number of minutes until the caller's next appointment and the expected duration of the call, respectively.

4.1.1.4. Ring Controller

The ring controller is responsible for the ring thread and also for determining which ring tone to use given a particular scenario and user context. There are three types of ring tones built into inCall: the sound of someone snoring, the sound of a monkey screaming and the sound of a regular digital telephone. The monkey screaming tone is used to convey urgency or speed, such as when the caller should return a phone call very soon in order to talk to the callee. This might happen if the callee will be in a meeting or away shortly. The sound of someone snoring is used in the exactly opposite situation, such as when the caller does not need to return a phone call right away or when the expected duration of a call is long. The sound of a regular digital telephone system is used in the in-between cases. The tables below show the relation between time availability, expected call duration and ring tones chosen by inCall.

Time Availability	**Ring Tone**
less than 30 minutes	Monkey
between 30 minutes and 2 hours	Digital Telephone
more than 2 hours	Snore

Table 3: Time Availability vs. Ring Tone

Expected Call Duration	**Ring Tone**
less than 10 minutes	Monkey
between 10 minutes and 30 minutes	Digital Telephone
more than 30 minutes	Snore

Table 4: Expected Call Duration vs. Ring Tone

4.1.2. Services

4.1.2.1. Availability of caller encoded in phone ring frequency

1. The caller places a call to the recipient by entering the 'call' command as a command line. This requires an IP address and an identifier for the recipient. This starts up the call thread and puts the phone in the calling state.

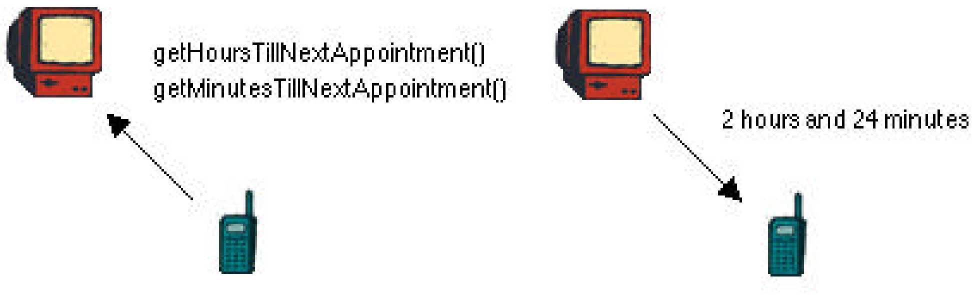

getHoursTillNextAppointment()
getMinutesTillNextAppointment()
2 hours and 24 minutes

Figure 4: Call thread requests caller's time availability to caller's computer

2. The call thread contacts Sensorama in the caller's personal computer to get the context data: time until next event (*getHoursTillNextAppointment* and *getMinutesTillNextAppointment*). It then contacts the recipient's phone with a *CallRequest* message containing the addresses of the caller's phone and desktop, and all the context data.

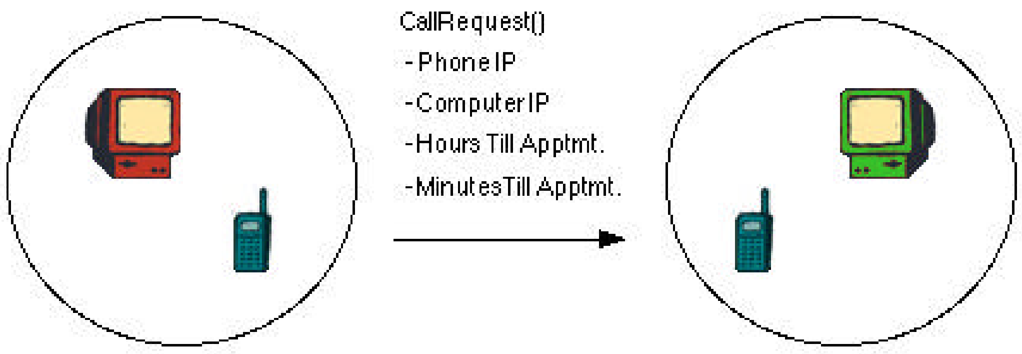


Figure 5: Call thread contacts the recipient's phone

3. Provided that the *CallRequest* returns true (i.e. the recipient is not busy), the call thread goes into a waiting loop. Unless it's stopped (by the call being answered), it times out and returns the phone to an idle state in 35 seconds.

4. The recipient's phone receives the *CallRequest* message through its XML-RPC server. If it's not busy, it stores the data about the incoming call (IP addresses and context data) in a global structure, and starts up the ringer thread.

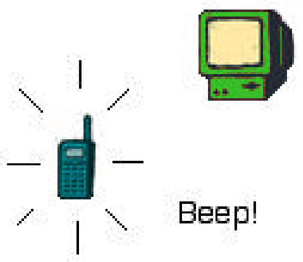


Figure 6: Ring thread selects a ring tone

5. The ring thread uses the context data and internal logic to decide what ring tone to use. It then starts a loop that lasts until it's stopped (by the answer command) or until it times

out in 30 seconds. During this loop it plays the selected sound by forking off a sub process and executing the player process (SOX, in this case [21]).

6. While the ring thread is running, the recipient’s phone sends the context information to the recipient’s computer, by issuing a *CreateSensorWindow* message.

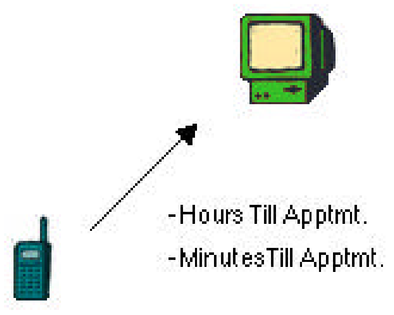


Figure 7: Recipient's phone sends context data to recipient's computer

7. The recipient’s computer then displays the context information to the recipient of the call in a *SensorWindow*.

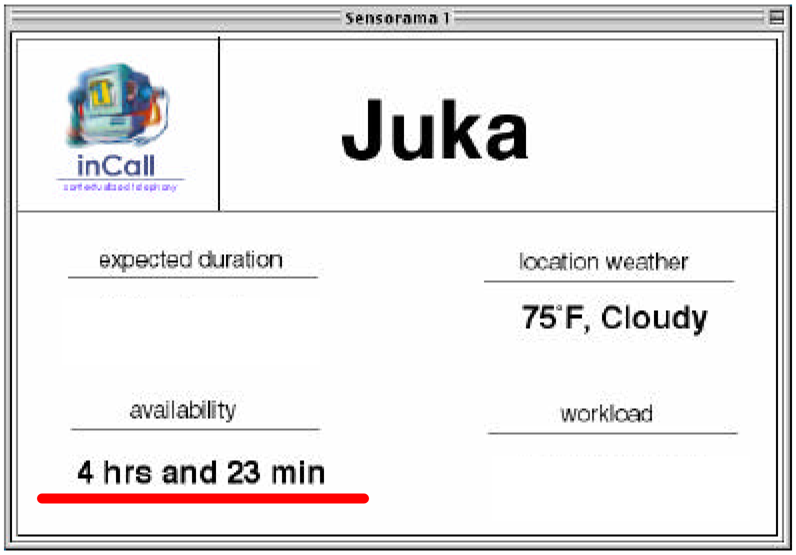


Figure 8: Recipient's computer displays caller's context data

8. The recipient chooses to answer the ringing phone by invoking the answer command. This shuts down the ring thread and starts up the full duplex thread. The full duplex thread begins sending audio and prepares to receive incoming audio. Finally, a *CallAnswered* message gets sent back.

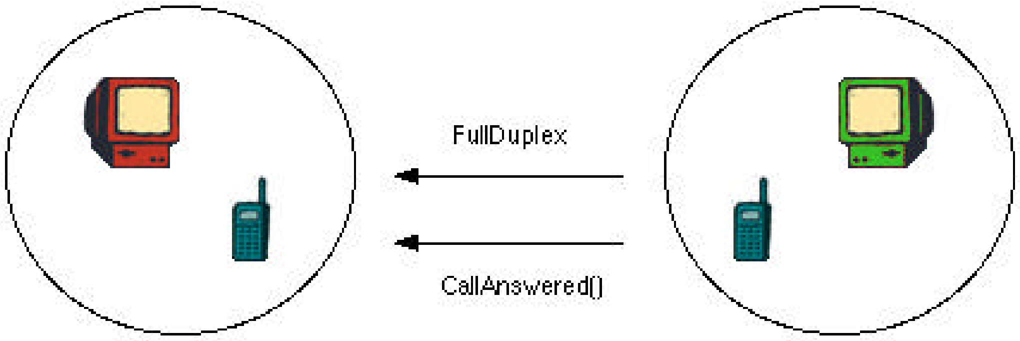


Figure 9: Recipient's phone agrees to start full-duplex

9. The *CallAnswered* message is received by the caller's XML-RPC server. It stops the call thread, and starts up the fullduplex thread. This begins full-duplex audio exchange between the two parties.

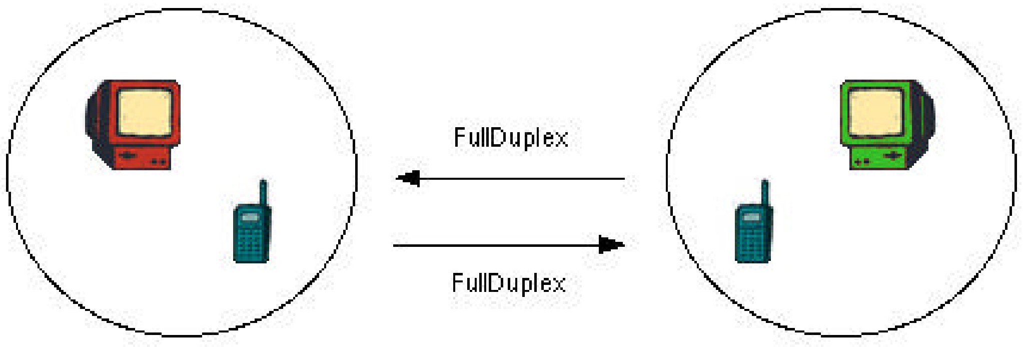


Figure 10: Full-Duplex starts on both ends

10. Once the conversation is finished either side can use the close command, which stops the full duplex thread.

11. The other side's full duplex process, having not detected any incoming data in a certain period (5 seconds), will time out and shut down. The phone will then be returned to the idle state. This is also what happens if the connection is dropped.

4.1.2.2. Expected duration of a call encoded in phone ring or frequency

1. The caller places a call to the recipient by entering the 'call' command as a command line. This requires an IP address and an identifier for the recipient. This starts up the call thread and puts the phone in the calling state.

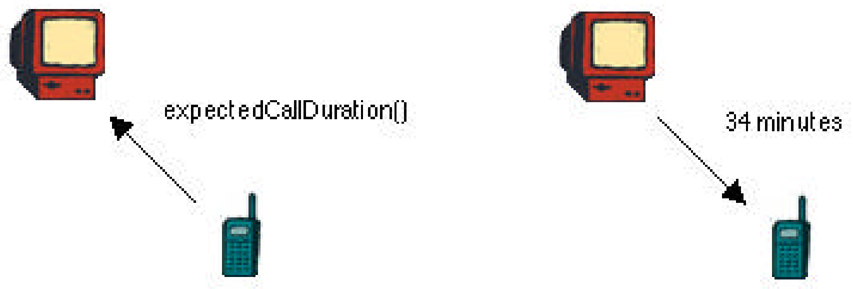


Figure 11: Call thread requests expected call duration to caller's computer

2. The call thread contacts Sensorama in the caller's personal computer to get the context data: expected call duration (get*ExpectedCallDuration()*). It then contacts the recipient's phone with a *CallRequest* message containing the addresses of the caller's phone and desktop, and all the context data.

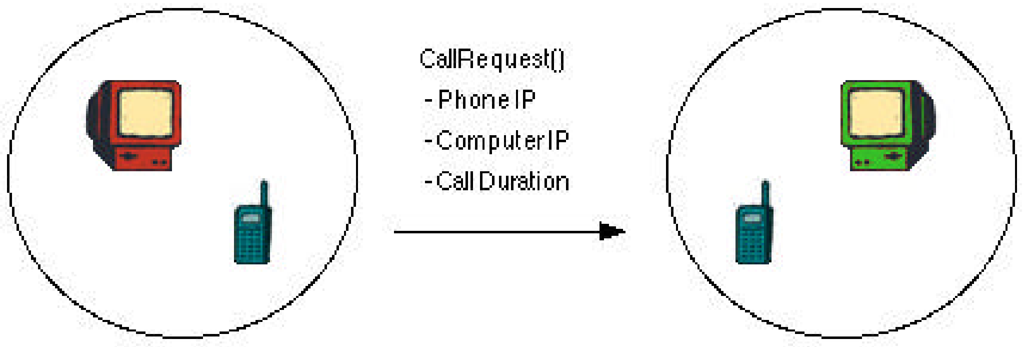


Figure 12: Call thread contacts the recipient's phone

3. Provided that the *CallRequest* returns true (i.e. the recipient is not busy), the call thread goes into a waiting loop. Unless it's stopped (by the call being answered), it times out and returns the phone to an idle state in 35 seconds.

4. The recipient's phone receives the *CallRequest* message through its XML-RPC server. If it's not busy, it stores the data about the incoming call (IP addresses and context data) in a global structure, and starts up the ringer thread.

Figure 13: Ring thread selects a ring tone

5. The ring thread uses the context data and internal logic to decide what ring tone to use. It then starts a loop that lasts until it's stopped (by the answer command) or until it times out in 30 seconds. During this loop it plays the selected sound by forking off a sub process and executing the player process (SOX, in this case [21]).

6. While the ring thread is running, the recipient's phone sends the context information to the recipient's computer, by issuing a *CreateSensorWindow* message.

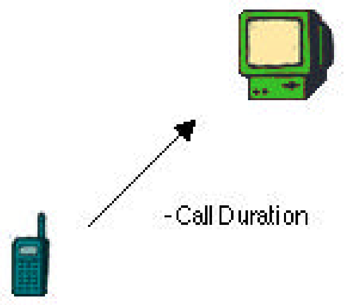
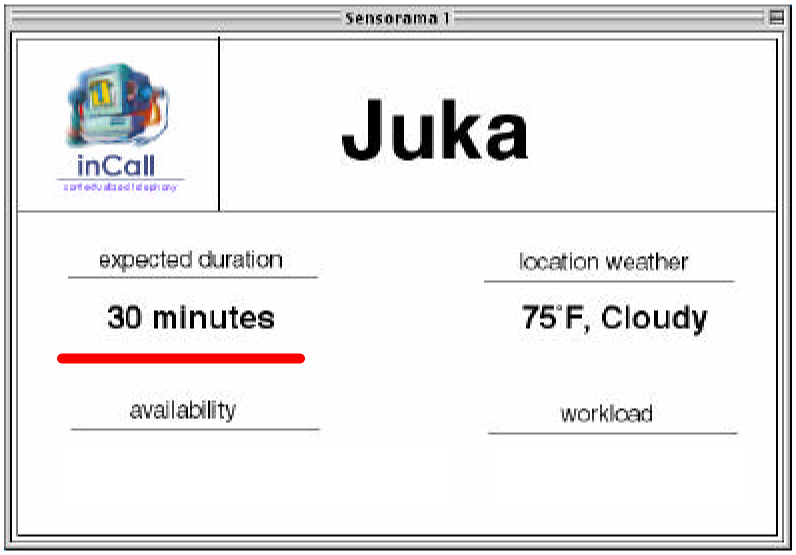


Figure 14: Context data sent to recipient's computer

7. The recipient's computer then displays the context information to the recipient of the call in a *SensorWindow.*



Juka

expected duration

30 minutes

location weather

75°F, Cloudy

availability

workload

Figure 15: Recipient's computer displays caller's context data

8. The recipient chooses to answer the ringing phone by invoking the answer command. This shuts down the ring thread and starts up the full duplex thread. The full duplex thread begins sending audio and prepares to receive incoming audio. Finally, a *CallAnswered* message gets sent back.

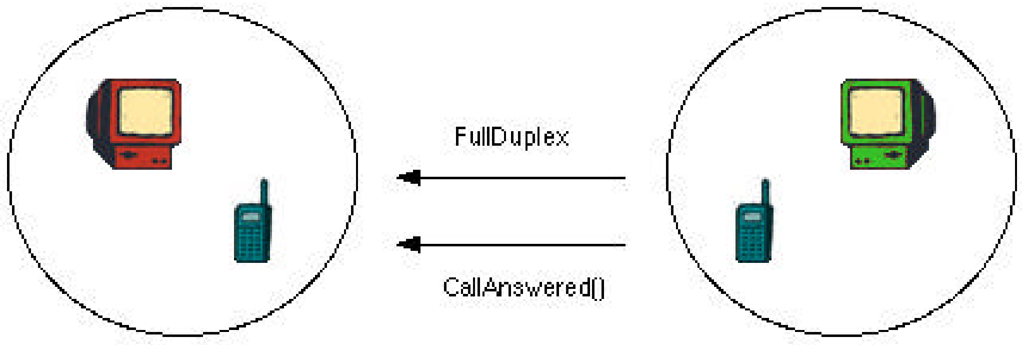


Figure 16: Recipient's phone agrees to start Full-Duplex

9. The *CallAnswered* message is received by the caller's XML-RPC server. It stops the call thread, and starts up the full duplex thread. This begins full-duplex audio exchange between the two parties.

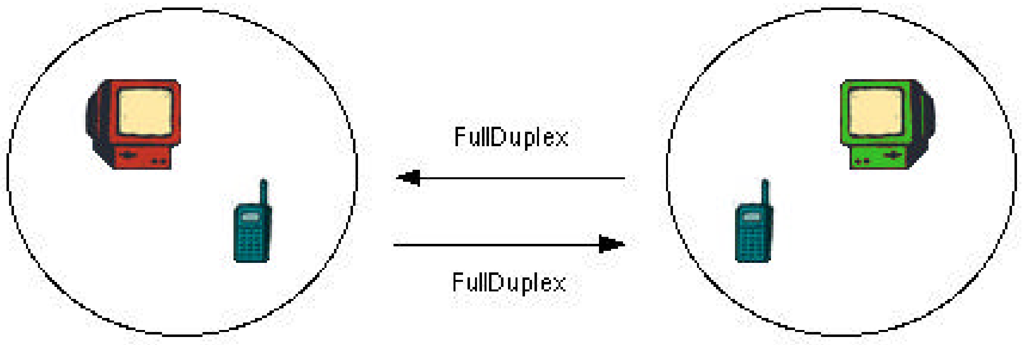


Figure 17: Full-Duplex starts on both ends

10. Once the conversation is finished either side can use the close command, which stops the full duplex thread.

11. The other side's full duplex process, having not detected any incoming data in a certain period (5 seconds), will time out and shut down. The phone will then be returned to the idle state. This is also what happens if the connection is dropped.

4.2. Sensorama

Sensorama is an application and interface that allows a computer personal information manager to become sensible and exchange user context information with other devices, such as a telephone. In reality, Sensorama was implemented as a wrapper around Microsoft Entourage for the Macintosh, a software package that includes an email client, a calendar and also an address book. Sensorama constantly monitors the data activity of Entourage, such as the flow of email messages and appointments of users, and either tries to analyze it or make some of it available to other systems and applications as a set of distributed observable interfaces [23]. Sensorama is a core element of the service because it is the source of user context information to inCall. The telephone component of inCall simply needs to comply with the XML-RPC interface of Sensorama in order to request information from it. Examples of what kind of personal information Sensorama provides includes how long users will be free until their next appointment and whom they communicate with by email regularly.

One of the challenges of an application like Sensorama has to do with the Informativeness vs. Privacy design tradeoff mentioned by Milewski and Smith [4]. Milewski and Smith state that if someone’s personal information is conveyed fully enough to be useful, in the case of a telephone awareness application, then it may often violate that person’s privacy. Naturally, since I am proposing a system that promotes the flow of personal information among devices, there is no question that privacy and the extent to which users are willing to give it up is one of the major challenges that inCall faces. Reducing the resolution of information to address privacy concerns has been one of the tactics employed in the past in dealing with this compromise and this is the strategy that I use here as well [6]. Although the exposure of private user information to the external world might be undesirable at first thought, it should not pose any problems or threats to users as long as all sensitive information is kept within a closed and personal computational environment.

Even though Sensorama was described as a wrapper around Entourage, it was fully implemented as a Macintosh stand-alone application. The next several sections offer a detailed description of the Sensorama interfaces, how they were implemented and other important elements of this non-trivial piece of inCall.

4.2.1. Interfaces and Organization

The Sensorama interfaces are constituted of modular managers organized in layers. At the bottom lays the low-level manager, the Activity Manager, whose functionality provides essential services to higher-level managers. Some high-level managers examine the raw data provided by the Activity Manager, extract regularities from it and expose them with an interface to the outside world. Others, such as the Location Manager, simply offer information services and do not need to communicate with the Activity Manager at all. The regularities extracted from the Activity Manager data might be patterns of social interaction or time availability, for example. There are three high-level managers, the Time Manager, the Location Manager and the Social Manager. Below is a diagram of the Sensorama managers in-context:

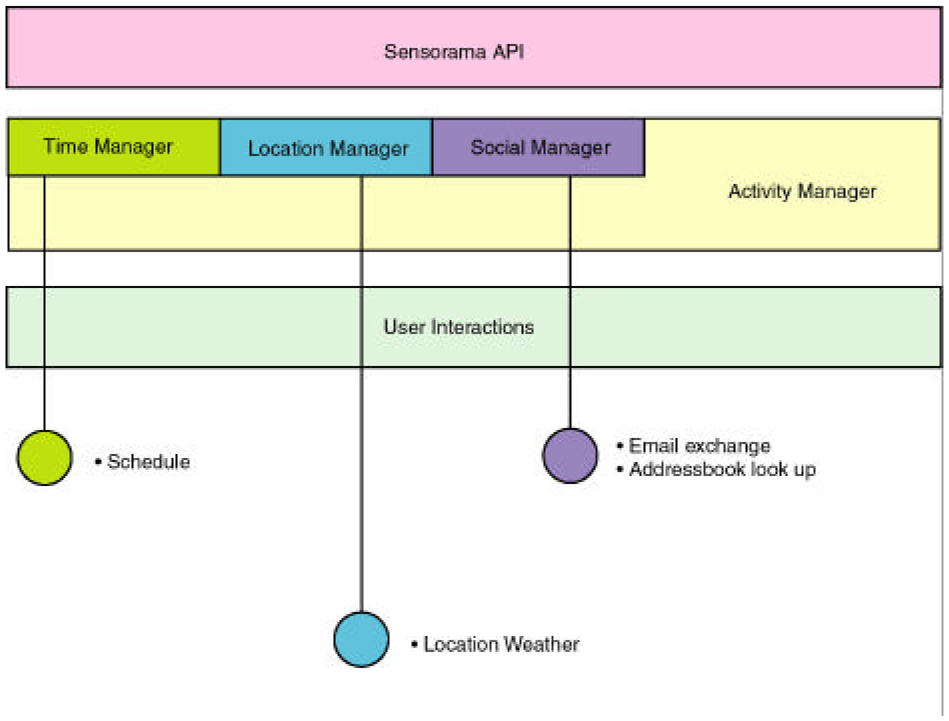


Figure 18: Sensorama architecture

4.2.1.1. High-Level Managers

4.2.1.1.1. Time Manager

The Time Manager is responsible for collecting scheduling information about the user from a personal information manager and sending this piece of context data to inCall. As it currently stands, this manager is very simple, since it is not trying to make any direct inferences about the state of the user.

Integer	getHoursTillNextAppointment()
Integer	getMinutesTillNextAppointment()

Table 5: Time Manager API

The two interfaces, *getHoursTillNextAppointment* and *getMinutesTillNextAppointment* both return an Integer indicating the number of hours and minutes respectively, until the next user appointment. A much larger set of interfaces could be implemented for the Time Manager, but the two described above were the only ones strictly necessary for the implementation of the inCall user scenarios.

Incidentally, the Appendix B describes a method for determining user workload that would be a possible way to extend the Time Manager.

4.2.1.1.2. Social Manager

Recognizing email exchange patterns is the main task of the Social Manager. It provides two interfaces, *isPersonAContact* and *getExpectedCallDuration*, which mainly keep track of user contacts and communication history. While the *isPersonAContact* interface simply checks whether a contact can be found in the user's addressbook, the *getExpectedCallDuration* interface is more complex. It was added to Sensorama in order to implement one of inCall's user scenarios. The *getExpectedCallDuration* interface deduces how long a call between A and B might last according to the number of emails that A has received from B or vice-versa. For instance, if A has received many emails from B, in other words, is in constant communication with B, then a phone call between the two of them is likely to be relatively short. Alternatively, if A has received only one email from B in a long time, then a call from A to B is probably going to be longer. One of the challenges with this interface is that intuitively, the rationale behind its implementation might make sense. However, in practice, the exact opposite of what we might expect could naturally occur. The *getExpectedCallDuration* interface correlates email number with call duration according to the table below:

Number of Emails	**Expected Call Duration**
> 4	5 minutes
> 1	20 minutes
1	35 minutes

Table 6: Email Number vs. Call Duration

Due to time constraints, the effectiveness of this interface has not been evaluated. It is also worth noting that the Social Manager does not try to make inferences based on the semantics of a communication stream, such as the contents of an email message. Here in the MIT Media Laboratory, Andrea Lockerd is currently working on a much more sophisticated system whose goal is to extract emotional resources from the content of email messages to infer the types of social connections that an individual has with others.

Boolean	isPersonAContact(StringName)
Integer	getExpectedCallDuration()

Table 7: Social Manager API

4.2.1.1.3. Location Manager

The Location Manager provides information about the user environment, more specifically weather temperature, by querying an external source: the web site of The Weather Channel. Given a zip code, this manager issues a URL request to the following URL, with the zip code appended as shown below:

http://www.weather.com/weather/local/ + zip code

Table 8: The Weather Channel URL

The returned web page is then parsed and the temperature is obtained. It is important to mention that the web page above was designed to be accessible by web browsers, so its layout may change at any time and the parsing code implemented by the Location Manager may stop functioning correctly. Ideally, a web service with a permanent interface would be the provider of this type of user context information.

Integer	getLocationTemperature()

Table 9: Location Manager API

4.2.1.2. Low-Level Managers and API

4.2.1.2.1. Activity Manager

The Activity Manager is a central piece of Sensorama. It is responsible for constantly monitoring the activity and data of the user's personal information manager. Contextual observations harvested by the Activity Manager are then sent to high-level managers to be analyzed and from where regularities are extracted. With regard to communication with the personal information manager application, Sensorama relies on the interprocess communication interface built into the Macintosh operating system API. This interaction is described in detail in a later section.

4.2.1.3. Communication Protocol

The Sensorama interface is exposed through XML-RPC [40], a simple but effective wire protocol built on top of HTTP [42]. In other words, XML-RPC is the interface between Sensorama and the telephone component of inCall. The word XML-RPC is a combination of two different acronyms: XML and RPC. XML stands for "eXtended Markup Language" and RPC stands for "Remote Procedure Call". In essence, XML-RPC communicates using XML in order to perform remote procedure calls.

A distributed protocol like XML-RPC was chosen as the Sensorama interface primarily because of its simplicity, reliance on XML and widespread support due to recent interest in the development of web services on the Internet [7]. One of the great advantages of an HTTP-based protocol is that it allows systems running on disparate operating systems and environments to communicate with each other rather seamlessly. Any device, using any operating system connected to a network can in principle implement an XML-RPC client and server and become part of a distributed network. An XML-RPC call to the a hypothetical ***calculateArea*** interface would look like this:

```
<?xml version="1.0"?>
<methodCall>
          <methodName>calculateArea</methodName>
          <params>
                    <param>
                              <value><string>Triangle</string></value>
                    </param>
          </params>
</methodCall>
```

Table 10: Sample XML-RPC Call

The document above is formatted in XML and would be sent from a client to a server as a POST request. A typical reply would look like this:

```
<?xml version="1.0"?>
<methodResponse>
          <params>
                    <param>
                              <value><i4>36 </i4></value>
                    </param>
          </params>
</methodResponse>
```

Table 11: Sample XML-RPC Response

In particular, an XML-RPC call to the ***getMinutesTillNextAppointment*** interface of Sensorama would look like this, with the header included:

```
POST /RPC2 HTTP/1.0
User-Agent: xmlrpclib.py/0.9.8 (by www.pythonware.com)
Host: 18.85.19.57:8080
Content-Type: text/xml
Content-length: 113

<?xml version="1.0"?>
<methodCall>
        <methodName> getMinutesTillNextAppointment </methodName>
                <params>
                </params>
</methodCall>
```

Table 12: Sample XML-RPC Call with Header

And if there are ***38*** minutes remaining until the next user appointment, Sensorama returns the following message, also with the header included:

```
HTTP/1.0 200 OK
Connection: close
Content-Length: 112
Content-Type: text/xml
Date: Mon, 1 April 2002 12:45:00 GMT
Server: Sensorama MacOS Server

<?xml version="1.0"?>
<methodResponse>
      <params>
            <param>
                  <value><i4>38</i4></value>
            </param>
      </params>
</methodResponse>
```

Table 13: Sample XML-RPC Response with Header

This last example is very important because it accurately illustrates how an interaction with Sensorama takes place with XML-RPC. The fact that XML-RPC is based on XML makes client and server messages easy to read and debug.

4.2.2. Implementation

The MacOS implementation was designed as a stand-alone application. It is divided in seven modules: MacZoop, XML Parser, XML-RPC Objects, Sensorama Core and API, IPC, XML-RPC Client and XML-RPC Server. The following diagram illustrates how these modules are organized:

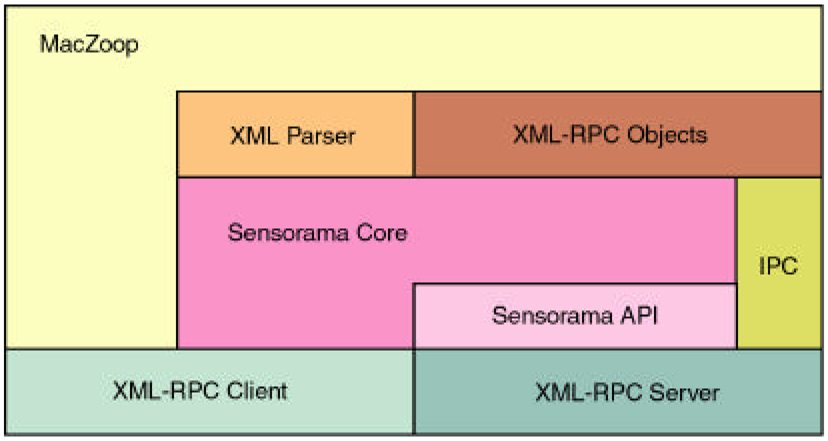


Figure 19: MacOS Sensorama implementation

4.2.2.1. MacZoop

Modern operating systems with increasingly complex user interfaces such as the MacOS and Windows pose a challenge to software developers. To facilitate the development of software programs with sophisticated user interfaces, a variety of GUI toolkits and frameworks emerged in the last decade, such as MacZoop. MacZoop is an object-oriented framework that makes it really simple to design software applications for the MacOS [35]. Most of the essential routines that handle the management of the user interface are already part of the framework; in other words, it does not need to be written by software programmers. By management of the user interface I am referring to tasks such as sending and receiving events to and from the operating system, handling mouse clicks in menus and controls and supporting "drag & drop" in windows. MacZoop is the framework on top of which I built Sensorama for the MacOS.

4.2.2.2. XML Parser

The XML Parser of Sensorama is based on Expat, a library, written in C, for parsing XML documents [36]. It's the underlying XML parser for many open-source XML parsers currently distributed in a variety of formats. It is very fast and sets a high standard for reliability, robustness and correctness. However, it is not a validating parser, in other words, the documents that it parses must be well-formed XML documents but not necessarily bound to any DTD. This library is the creation of James Clark, who was also the technical lead on the XML Working Group at W3 that produced the XML specification.

Expat is a stream-oriented parser. Handler functions need to be registered with the parser before it starts processing a document. As the parser recognizes parts of the document, it calls the appropriate handler for that part. In Sensorama, Expat was encapsulated in an object to facilitate its integration with the rest of the object-oriented architecture in place. The body of the *parse* method of the *XMLParser* class is shown below:

```
XML_Parser parser = XML_ParserCreate(NULL);
XML_SetUserData(parser, this);

// Create a pointer to the static method startElement_static
void (*startElement_static)(void *, const char*, const char**);
startElement_static = &XMLParser::startElement_static;

// Create a pointer to the static method endElement_static
void (*endElement_static)(void *, const char*);
endElement_static = &XMLParser::endElement_static;

// Create a pointer to the static method textElement_static
 void (*textElement_static)(void *, const char*, int);
textElement_static = &XMLParser::textElement_static;

// Set the handlers
XML_SetElementHandler(parser, startElement_static, endElement_static);
XML_SetCharacterDataHandler(parser,textElement_static);

// Parse the document
XML_Parse(parser, buf, len, true);

// Delete Parser
XML_ParserFree(parser);
```

Table 14: Using the Expat XML Parser

In the first line, an Expat instance is created. ***StartElement_static***, ***endElement_static*** and ***textElement_static*** are set to point to the parser handlers. Later, calls to ***XML_SetElementHandler*** and ***XML_SetCharacterDataHandler*** with the parser handlers let Expat know which callbacks it should use when it encounters an opening XML tag, a closing XML tag and the text enclosed by the two previous tags, respectively.

4.2.2.3. XML-RPC Objects

In order to format data back and forth between native C types and XML-RPC types, I designed a set of classes that encompass a simple but effective object model for XML-RPC. This object model shares many similarities with the Document Object Model (DOM) specification of the W3C [43], but it is much more specific to XML-RPC, naturally. When creating new messages to be transmitted over the XML-RPC interface, these classes facilitate the programmatic composition of XML tags and data. Likewise, when extracting data from an XML-RPC message, the classes return an object model that we can manipulate fairly easy from C and C++. An example, if we were calling the interface ***isPersonAContact*** of the Social Manager located at 18.85.19.165 with the string argument "***Mark Whitman***", this is how we would do it:

```
// Create a new method object and give it a name
XMLRPC_Method method;
method.setName("isPersonAContact");

// Create a string value and give it a value
XMLRPC_string_Value value;
value.setValue("Mark Whitman");

// Add the string object to the list of  values of the method object
method.addToValueList(&value);

// Print the XML-RPC message and send the request
char* msg = new char[2048];
method.print(msg);
Handle result = aClient->postXMLRPCRequest("http://18.85.19.165", msg);
```

Table 15: Sample Usage of XML-RPC Objects

The illustrative code above is self-explanatory. An ***XMLRPC_Method*** object is created and given a name, ***isPersonAContact***. An ***XMLRPC_string_Value*** is also instantiated and set to value '***Mark Whitman***". The string value object is then added to the list of values of the method object. The msg buffer contains the XML below before the postXMLRPCRequest message is called:

```
<?xml version="1.0"?>
<methodCall>
        <methodName>isPersonAContact</methodName>
        <params>
                <param>
                        <value><string>Mark Whitman</string></value>
                </param>
        </params>
</methodCall>
```

Table 16: Output of Sample Usage of XML-RPC Objects

Incidentally, the XML Parser was designed to work very closely with the XML-RPC Objects. In fact, what the Sensorama XML Parser outputs is an object tree based entirely on XML-RPC Objects. Below is the complete set of XML-RPC Objects:

```
XMLRPC_Method
XMLRPC_array_Value
XMLRPC_boolean_Value
XMLRPC_i4_Value
XMLRPC_string_Value
XMLRPC_struct_Value
XMLRPC_Member
```

Table 17: List of XML-RPC Objects

4.2.2.4. Sensorama Core and API

The Sensorama Core and API is the main module of the application, the heart of the Sensorama implementation. The class *Sensorama* is the one that implements the Sensorama interface. To a very large degree, all the other modules such as the XML-RPC Client and Server exist to support the *Sensorama* class.

The class itself is organized in different sets of methods, a set for each different Sensorama manager. Additionally, there is a *dispatchMethod* procedure, which simply calls the right Sensorama method by name. Most of the methods depend on the interprocess communication facility of the MacOS to communicate with other applications. How exactly Sensorama relies on the Interprocess Communication and Scripting (IPC) architecture of the MacOS is described in the next section.

It is important to say that there is only one Sensorama object, which gets created when the Sensorama application first starts. The Sensorama class was designed following the Singleton design pattern in order to enforce this requirement [22]. The only other module that communicates with the Sensorama object during the execution of Sensorama is the XML-RPC Server.

4.2.2.5. Interprocess Communication and Scripting (IPC)

The MacOS implementation of Sensorama was created as a stand-alone application. There is one particular reason why this design decision was made: interprocess communication (IPC). The interprocess communication facilities of the MacOS, it is fairly easy to allow completely independent applications to communicate with each other and exchange information. In particular, I am referring to Apple Events, Applescript and the Open Scripting Architecture (OSA) [45]. All Macintosh applications are required to implement an API for interprocess communication, according to guidelines released by Apple Computer. While not all applications do so, especially the ones designed by small software developers, most of the major software packages for the MacOS provide an interface for IPC.

Interprocess communication is important to Sensorama because two of its high-level managers, the Time Manager and the Social Manager, both need to obtain data found in the personal information manager and email client application of users, such as user's schedule for the day or emails sent in the last 24 hours. As mentioned in the Sensorama spec above, the Time Manager and the Social Manager do not interact with the user environment directly. The Activities Manager is the one that retrieves information from user applications. Regardless, the need to access information found in these user applications exists and must be fulfilled to implement Sensorama. Applescript enables a simple script asking for an upcoming user appointment to easily retrieve that information from Microsoft Outlook. As an example, the script below is used to retrieve all events from a user's calendar in Microsoft Entourage:

```
tell application "Microsoft Entourage"

     set eventsList to {}
     set theEvents to every event

     repeat with oneEvent in theEvents
          set event_subject to subject of oneEvent
          copy event_subject to the end of eventsList
     end repeat

end tell

set result to eventsList
```

Table 18: Sample AppleScript

Another reason why using scripts is useful in this case is because not everyone uses Microsoft Outlook as their information manager. In fact, there are many others popular personal information managers in the market and we would like Sensorama to work with all of them, or at least with as many as possible. To make the script above work with another application, say Now Up-To-Date & Contact [46] and keep the rest of the Sensorama implementation absolutely intact, all that is required to change the very first line, from ‘`tell application "Microsoft Entourage"`’ to ‘`tell application "Now Up-To-Date"`’. Such a change can be made to not even need source code recompilation. This is one of the reasons why I claimed that the MacOS implementation of Sensorama is reusable and multi-purpose.

Usually, on the Macintosh, stand-alone applications are implemented in languages such as C and C++ and they don’t interface directly with scripts such the one shown above. In order to communicate with other running applications, most programs rely on Apple Events, which is a C interface to the interprocess communication architecture available on the MacOS. The problem with the Apple Events interface is that it is not easy to use, especially when more complex data exchanges are needed between applications. Multiple descriptors have to be created and inserted into each other, sometimes in the form of lists. The end result is code that is hard to maintain and understand. Nevertheless, thanks to the Open Scripting Architecture (OSA), it is also possible to execute and control natural language scripts such as Applescript from C code. According to Apple, this is the definition of the OSA:

“The Open Scripting Architecture (OSA) is an API that provides a standard mechanism for creating scriptable applications and for writing scripting components to implement scripting languages. It is made of a set of scripting component data structures, functions, and resources that allow applications to interact with scripting components.”

In summary, because of its simplicity, I decided to take the OSA path in order to make the Sensorama application communicate with other applications. The stand-alone app communicates with other external applications by means of AppleScripts, primarily.

4.2.2.6. XML-RPC Client

The XML-RPC client of the MacOS implementation of Sensorama is built around another Apple software technology, the URL Access Manager [25]. The URL Access Manager provides application support for downloading data from or uploading data to a Universal Resource Locator (URL). The client was encapsulated in a class called *HTTPSuperClient*, whose most important method is *postXMLRPCRequest*. Before *postXMLRPCRequest* is called, the client must always check whether the URL Access Manager is available by calling the method *urlAccessIsAvailable. PostXMLRPCRequest* takes two arguments; a URL and an XML-RPC method call in the form of an XML message. The method creates the XML-RPC request header and appends it to the XML message, as a prefix. The final step that *postXMLRPCRequest* is responsible for is to submit the combined header-message to the URL specified in the incoming argument as a POST. A typical instantiation and use of the *HTTPSuperClient* would look like this:

```
// Creates a new client
HTTPSuperClient* aClient = new HTTPSuperClient();

// Makes sure that the URL Access Manager is available
if(aClient->urlAccessIsAvailable())
{
        // Send the XML-RPC Message
        Handle result  = aClient>postXMLRPCRequest("http://18.85.19.165", msg);
}
```

Table 19: Using the XML-RPC Client

4.2.2.7. XML-RPC Server

The MacOS Sensorama XML-RPC Server runs in its own thread and spawns new threads for every incoming request, so the server is quite efficient. The MacOS Classic threading model and OpenTransport, the MacOS networking architecture are the two limiting factors that can cause bottlenecks to occur on the server side. However, unlike web servers, which must serve a large number of images and documents in a short period of time, the XML-RPC Server is not expected to handle a heavy load.

For every incoming XML-RPC incoming request, the code below gets executed. A description follows:

```
// The XML Parser Callback for XML-RPC
XMLParserCallbackXMLRPC* xmp = new XMLParserCallbackXMLRPC();

// Create an XML parser now
XMLParser* theXMLParser = new XMLParser(*xmp);

// Parse the XML now
theXMLParser->parseXML(xmlMessage, strlen(xmlMessage));

// Pointer to the request method object
XMLRPC_Method* r = theXMLParserForXMLRPC->getRequestMethodObject();

// Finally, execute the Sensorama method
XMLRPC_Method* theResponseObject = Sensorama::Instance()->dispatchMethod(r);

// The XML-RPC response is added here
if(theResponseObject)
        theResponseObject->printResponse(responseBuffer);
```

Table 20: XML-RPC Server Dispatching Calls

When an XML-RPC server request comes in, a new XMLParser parser is instantiated. The constructor of the parser takes an object of type XMLParserCallbackXMLRPC, which has all the required Expat handler functions. It is worth noting that the handler functions called when the parsing is taking place already format the incoming request with XML-RPC Objects and save it as an XMLRPC_Method object. The server then calls the Sensorama dispatchMethod with it, which returns an object of type XMLRPC_Method again, this time with the result properly formatted to be sent back to the client and originator of the request.

5. Evaluation

5.1. Description

The evaluation portion of this thesis is dedicated to inCall. More particularly, it focused on how extra context information provided to those receiving phone calls, specifically caller time availability, facilitates the job of contacting other people by phone. The hypothesis of this evaluation was that the exchange of time availability between phone systems could help eliminate what is commonly referred to as “phone tagging”; the definition of what happens when two or more people successively try and fail to contact each other by phone. The elimination of “phone tagging” is desirable because not being able to talk to someone when the need arises might delay the execution of tasks and the exchange of information, especially in a business setting.

To evaluate inCall, a comparative evaluation was organized with ten pairs of people in two studies. The people who participated in the studies were students at the MIT Media Lab. The goal of the first study was to test the evaluation procedures and also the inCall system interface. Essentially, the first study was a pilot test while the second study was the actual user evaluation. At first, I planned to organize a *qualitative* study to measure the benefits brought in by inCall, but with help from Ted Selker, one of my readers, I designed a user scenario that allowed me to test inCall *quantitatively.*

In pairs and in separate locations, users were asked to simulate a real world interaction between a travel agent and a travel magazine journalist pretending to be a customer trying to find last minute tickets, flight and accommodation to the World Cup Finals in Korea and Japan. In the scenario, the job of the travel agent was to take the customer call, search the web for travel deals and information relative to the customer requests and then phone the customer back with the travel offerings available. The job of the travel magazine journalist was to pretend to be a customer in order to evaluate the quality of service of the travel agency. The reason why this scenario was chosen is because it required both the agent and the fake customer to communicate with each other repeatedly over a relatively short period of time, increasing the chance that they would "miss" each other and that "phone tagging" would occur. Both the journalist and the travel agent were given schedules to constrain their communication and simulate meetings and appointments during the study. The study was set up to last approximately 30 minutes and the metric adopted to evaluate the performance of inCall was the ratio of the total number of calls received divided by the total number of phone calls made. The time spent for the completion of tasks described in the user study guidelines was also taken into consideration, together with the number of tasks completed. In simple terms, what the evaluation aimed to show was how the time availability context information of inCall gives users a better idea of when to return calls to avoid "phone tagging" and, thus help users accomplish collaborative tasks faster and with less calls.

Half of the users in the evaluation used a version of inCall that provided extra context information with regard to incoming calls and half of the users performed the evaluation tasks without the context information provided by inCall.

5.2. User study guidelines

Below are the guidelines given to the pair of subjects that participated in the user study, one for the travel agent and one for the journalist.

5.2.1. Journalist

Hello,

This is your first assignment as writer and journalist of **Travel Smart** magazine.

This month, we are going to publish a World Cup Finals special section in our magazine and you are going to be in charge of it. The first thing that we need to do is cover all the different travel agencies that offer bargain packages to Japan and Korea. Some of them sell cheap packages but their customer service is absolutely terrible and we want to steer our readers away from them. One of the agencies, and the one I would like you to focus on for the moment, is **Slouch Travel**. Slouch has a bad reputation; they are known for being lazy and not getting back to customers. But they are supposed to be improving, so we are going to TEST them!

Pretending that you are going to the World Cup Finals with your family, follow the instructions below IN DETAIL:

1. Call Slouch Travel and say: *"Hello, I am going to Japan next week to see the World Cup Finals. Can you give me the phone number of a Hilton Hotel in Tokyo?"*

2. When they call back with the phone number, say: *" I am sorry, but I have changed my mind. Can you try to find the phone number of a youth hostel in Tokyo instead?"*

3. Finally, the third time that they call, ask them for flight info. Say: *"How much would an airfare to Tokyo cost? I want to leave Boston June 25th and come back July 3rd. No preference for airline or flight time."*

4. To simulate a real person calling the travel agency, pretend that you have the schedule below and that the dark rectangles correspond to meetings. **IF THE PHONE RINGS DURING THOSE TIMES, DO NOT PICK UP THE PHONE. PRETEND YOU ARE NOT IN THE OFFICE.**

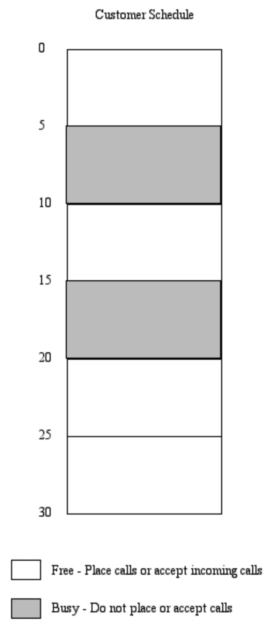


Figure 20: User Study: Journalist Schedule

5. Fill-out a travel agency evaluation form whenever the phone rings or whenever you place a customer call, regardless of whether you talk to the agent or not.

5.3.2. Travel Agency

Welcome to **Slouch Travel**, a travel agency specialized in low-cost vacation packages for people on a shoestring budget. This is your first day on the job. You are a travel operator and your function is to take phone calls from customers inquiring about the cost and availability of flights, hotels, cruises and get-away specials. Below are your job instructions:

1. At Slouch Travel, we only work 30 minutes a day, with several breaks in these 30 minutes. You can find your schedule below. **DO NOT WORK DURING YOUR BREAKS.**

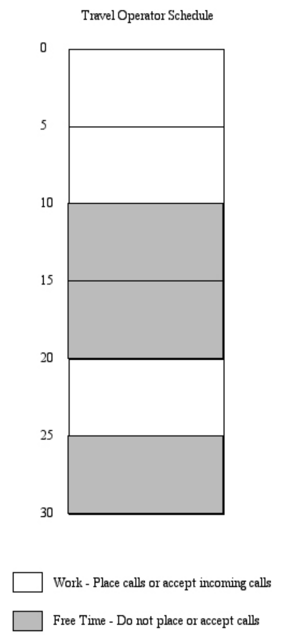


Figure 21: User Study - Travel Agent Schedule

2. When customers call with questions, **LISTEN TO THEM CAREFULLY** and tell them that you are **CALLING BACK AS SOON AS POSSIBLE WITH ANSWERS**. **DO NOT ASK ANY MORE QUESTIONS.**

3. The web is your information source and your friend. Go to the web and search for flights, hotels, cruises and packages for all of your customer requests.

4. Fill-out a customer interaction form whenever the phone rings or whenever you place a customer call, regardless of whether you talk to the customer or not. **IF THE PHONE RINGS DURING YOUR BREAK, DO NOT PICK UP THE PHONE. PRETEND THAT YOU ARE NOT IN THE OFFICE.**

5. **GET BACK TO CUSTOMERS AS QUICKLY AS POSSIBLE!**

5.3. Results

Overall, the evaluation of the inCall system demonstrated how time availability context information can help people perform collaborative tasks together. The first study was a great aid to debug the evaluation procedures and guidelines. As an example of a significant change that resulted from testing inCall, when inCall was originally designed, it provides two user interfaces for control: a command-line shell prompt and an XML-RPC server. Because I wanted to make it as natural as possible for users to place and answer calls, I designed a user-friendlier interface for inCall specifically for this user study, the inCall Caller. During the evaluation test run, I noticed that the Caller did not provide enough user feedback and negatively influenced how users perceived the inCall state to be. For the actual evaluation, users were asked to interact with inCall using the iTC command shell.

Four pairs of people were part of the evaluation pilot and six pairs of people were part of the real evaluation. During the two iterations of the real user evaluation, the total number of calls using the inCall system was 41.

In the first three iterations of the evaluation, users were given access to the time availability context information of each other, through the inCall system. The total number of calls was 19. Of these 19 calls, 9 of them led to a conversation between the travel agent and the journalist. The ratio of the number of calls received divided by the total number of calls made was 0.47. In other words, 47 percent of the calls placed resulted in a conversation between the pair of subjects. In total, 7 tasks were completed.

The last three iterations were marked by a larger number of calls placed, 22 and also by how few calls led to an interaction between the agent and the journalist, only 8. In this iteration, users were also using inCall to place calls, but they were not receiving time availability information when receiving calls. The ratio of the number of calls received divided by the total number of calls made was 0.36. In conclusion, only 36 percent of the calls made resulted in a conversation between the pair of subjects. This small call percentage illustrates a higher incidence of “phone tagging”. In total, 5 tasks were completed.

	Calls Made	**Calls Rcv**	**Tasks Completed**
inCall	19	9	7
Phone	22	8	5
Total	41	17	

Table 21: inCall vs. IP Phone Comparison for the Pairs

The results obtained in the six iterations of the user evaluation indicate that the exchange of time availability as a piece of context information between users in a communication setting seems to reduce the amount of “phone tagging”. However, because of how few pairs of subjects participated in the evaluation, more studies are necessary to establish statistical corroboration of time availability as a piece of context in communication. A study lasting days, if not weeks, might be ideal to really test the behavior of the unique inCall features in comparison with what a typical IP phone offers in functionality. This evaluation shows that the utilization of time availability as an element of context can certainly help people communicate with each other more efficiently.

6. Contributions

inCall is a communication service using voice over IP that shows how user context information acquired from a personal information manager can be utilized to help users prioritize incoming calls and make better informed decisions about them. The primary contributions of inCall are:

1. Showing how a voice over IP network offers the opportunity to serve as a dynamic and malleable foundation for new types of communication services, context-based or not, that can be integrated with voice services. In other words, inCall reiterates how many possibilities exist in an environment where voice and data can coexist interchangeably.

2. Demonstrating how devices that gather and share context information about how they are used can collaborate to create new user-centric communication services. Although simple at first, this contribution of inCall is very powerful. It precipitates the development of a whole new category of multi-device services that transcends the features that each device can offer individually.

One element that differs inCall from other telephony awareness applications which is definitely a contribution as well has to do with the flow of context information within a collection of distributed devices. While in other telephony awareness systems context information flows from the callee to the caller [4,20], the essence of inCall comes from the flow of user context information from the caller to the callee. Without a doubt, this is another unique characteristic of inCall:

The key idea behind InCall is that when user A places a call to user B, inCall communicates with user A's computer and collects observational context information about user A, such as how busy he or she is, where he or she is and who he or she communicates with on a regular basis. This information is then sent to user B so that user B can better prepare for the call or make a better informed decision about whether to *take the call or not.*

As far as Sensorama, a component of inCall, is concerned, its main contribution is to show how we can develop a wrapper around a typical software application, such as a personal information manager. This is an important contribution because with such a wrapper, we can collect and analyze previously inaccessible data that software applications manipulate and subsequently share it with other devices that comply with a common interface.

7. Future Work

An area that could also benefit tremendously from future work and that I would like to emphasize here is evaluation. Long term user testing is the only important element of this research that is conspicuously missing due to time constraints.

As far as inCall is concerned, below is a list of changes and implementations that would significantly enhance the power and reach of the system:

- Make it easy for users to call each other. Right now, users have to type the command '*call*' followed by an IP number and the name of the person who is being called. The interface is nothing more than a command shell. It is important to make the interface user-friendlier for evaluation purposes and also for real-world usage. A simple user interface was developed for the evaluation, but it is not complete and easy-to-use enough for long term application testing.

- The inCall device right now is very heavy, because of the dual pocket slot, the wireless card and the 1GB Microdrive. It would be useful to try to reduce the size of the device. An option might be to reduce the RAM memory footprint and also the required hard drive space. The XML-RPC Client and Server library require a lot of hard drive space and that is the reason why the Microdrive is used. Perhaps in a future version of inCall, another less storage space-hungry XML-RPC library could be used.

- At the moment, different ring types are built into the inCall device and cannot be changed. Users like customization, so it would be useful to let users upload new rings to the device and also select the purpose and meaning of each ring.

- It would be very desirable to connect inCall to the real phone system and experiment with it in the context of real phone conversations. A comprehensive evaluation could follow.

- The reliability of the system needs to be improved for long-term user testing. Long-term user testing would be very desirable as well, especially how people would change their behavior with the addition of inCall into their lives.

- inCall right now is a point-to-point phone system. It could become an audio conferencing tool without too many modifications. It would be interesting to study the behavior of inCall within a group, in real-time.

- The audio quality of inCall can be improved as well. InCall could be modified to scale up and down in terms of bandwidth availability in real-time. In order words, inCall should be able to sense the amount of bandwidth available and change its audio encoding accordingly.

- Security and authentication are required element in a system like inCall that I did not address in this thesis. In principle, when sensitive personal information is exchanged between devices and applications, all the data must be encrypted and secure.

- To close a connection, users need to type the command '*close*'. There should be another more intuitive way to disconnect the device. Perhaps the '*close*' command could be assigned to one of the iPAQ built-in keys. Likewise for the '*call*' command.

- The two application scenarios implemented for inCall involve two different types of context information – time availability and social communication awareness. Other kinds of user context information could be employed in the development of user services. As an example, when users of inCall receive phone calls, they could also be informed of previous email communication that they have had with the incoming caller. Moreover, the web site of the incoming caller or the incoming caller's company could also be presented. These are extra services that could be integrated with the current features of inCall fairly easily.

- The Location Manager of Sensorama currently provides weather temperature information given a zip code. It would be more interesting if temperature and other types of context information tied to a particular physical location, could be obtained given an IP address. In the case of networked devices with an assigned IP address, there are ways to convert an IP address into a physical location. This can be accomplished by linking the IP address to its domain name. Most the time, domain names can give us some insights about its location (i.e. my computer here at MIT is called thomaz.media.mit.edu. The 'edu' termination indicates that it is located in an educational institution in the United States).

Personally, as the introduction of this thesis makes obvious, I am enthusiastic about the hidden potential of sensible personal digital devices that communicate with each other and share user-centric information. Below is a list of areas of exploration that I also find worth pursuing:

- In my opinion, inCall demonstrates only a couple of context-based services that could be designed. The popularization of web services [2] and the Semantic Web [7] are also likely to result in a large number of Internet resources that might work together with inCall and similar systems, enriching their potential even further.

- From the perspective of developing systems with human like intelligence and common sense, I believe that the distributed approach that I employed in inCall could shed some light in how we could get closer to some real-world systems that achieve practical levels of autonomy and perhaps intelligence. Undeniably, when I first proposed the characterization of a set of personal digital devices as a collection of sensible systems, I was drawing from The Society of Mind ideas of Marvin Minsky [3]. In his seminal work, Minsky suggests that our intelligence comes from the combined effort of several different task-specific agencies in our brain and not from a one single element of intelligence. Likewise, instead of relying on a single source of intelligence control to recreate an intelligent system, we could try to gather resources from different entities, in this case personal digital devices, with different characteristics but that speak the same language, to design a distributed sort of intelligence. As devices become increasingly smaller, more powerful and closer to our lives, and us we can start calling them sensors and my belief in such a distributed form of intelligence could come closer to fruition that way. It would be useful to try to evaluate whether these assumptions are right. The potential benefits are immeasurable.

- A high-level programming language to let users develop new distributed cross-device services like inCall could also be of interest. Such a language would allow users to create customized applications and alerts very easily. Below are some examples of user applications, assuming programmatic control over all of these devices:

1. An alarm clock that wakes you up in the morning with the song that your close friends have been listening to the most for the past 2 months.

2. A digital camera that automatically sends your pictures to friends and family when you get home from vacation.

3. An answering machine that sends you email every time someone leaves you a message.

- The Sensorama component of inCall was developed with the goal of being an observable API for software applications using open standard protocols. Its API was defined to some degree to fulfill the needs of inCall, a telephony application based on user context information. It would be interesting to investigate whether the key ideas of Sensorama would help other developers create services similar to inCall. An evaluation and testing program from the point of view of application and service developers could be a good way to answer this question.

8. Appendix A: Sensorama API Specification

8.1. Time Manager

Integer	getHoursTillNextAppointment()
Integer	getMinutesTillNextAppointment()

Table 22: Time Manager API

8.2. Social Manager

Boolean	isPersonAContact(StringName)
Integer	getExpectedCallDuration()

Table 23: Social Manager API

8.3. Location Manager

Integer	getLocationTemperature()

Table 24: Location Manager API

9. Appendix B: Workload Detection

At the moment, the Time Manager of Sensorama simply extracts temporal data from a personal information manager and makes it available to other devices, such as the inCall telephone component, in the form of an interface. Here is a description of a method that could be used by the Time Manager to attempt to infer user workload by examining the schedule of users and their typical communication patterns. First, below are a few examples of what the workload assessment method that I describe here should be able to infer:

- If a user has 4 appointments everyday on average in her business organizer, and one day that number jumps to 6 or 7, the workload assessment method should be able to detect that there has been an increase in workload on the part of the user.

- If on average, a user receives 14 email messages every day and that number jumps to 35 one day, the workload assessment method should detect that there has been an increase in communication activity on the part of the user.

To determine how much work a user is engaged in, there needs to be a way to quantify workload. I suggest that a metric be used here. The metric system that I propose is based on assigning points according to user activities per day and the amount of time that these activities last. These activities can be strictly communication activities or not. Below is a chart with the suggested workload values per activity:

- 10 points for each upcoming appointment in the user's schedule * duration/min.
- 4 points for each phone call initiated * duration/min.
- 3 points for each phone call received * duration/min.
- 2 points for each email message sent * duration/min.
- 1 points for each email message received * duration/min.

Where duration/min is the number of minutes that the activity lasted.

Table 25: Metric to determine workload

As an example, if one day someone has 4 appointments lasting 1 hours each, answers 5 phone calls that last 20 minutes each and writes 3 email messages that take 5 minutes to compose, the workload of this person in this particular day will be:

Appointments= 4 * (10 (appointment/points) * 60 (minutes/appointment)) = 2400
Phone Calls = 5 * (3 (phone call received/points) * 20 (minutes/call)) = 300
Emails Sent = 3 * (2 (email sent/points) * 5 (minutes/compose email)) = 30

Workload = Appointment Points + Phone Call Points + Emails Sent Points = **2730**

Table 26: Determining a workload

I believe that such a metric can be a reasonable and useful way to extend the Time Manager. Determining how much work a user faces is a necessity when it comes to prioritizing tasks on behalf of users.

10. Appendix C: Evaluation Forms

Customer Interaction Form – Slouch Travel

Time of Call:		
Circle One:	Call Received	Call Made
Returning Call:	Yes	No
Talked to Customer:	Yes	No

Reason for Call:

Travel Agency Evaluation Form – Travel Smart Magazine

Time of Call:		
Circle One:	Call Received	Call Made
Returning Call:	Yes	No
Talked to Agent:	Yes	No

Reason for Call:

11. Appendix D: Compiling LibWWW and XML-RPC-C

- Install LibWWW first and then XML-RPC-C.

11.1. LibWWW

1. Download the libwww .gz file from the W3C web site.
2. Untar the package using the command: tar xzvf <libwww filename>.
3. Enter the libwww untar-ed directory.
4. Type: ./configure –prefix= <libwww installation path here, such as /usr/local>.
5. Type make.
6. Type make install.
7. Add the path of the libwww installation + /bin to the PATH env var.
8. Add the path of the libwww installation + /lib to the LD_RUN_PATH env var.
9. Add the path of the libwww installation + /lib to the LD_LIBRARY_PATH env var.

11.2. XML-RPC-C

1. Download the xml-rpc-c .gz file from the W3C web site.
2. Untar the package using the command: tar xzvf <xml-rpc-c filename>.
3. Enter the xml-rpc-c untar-ed directory.
4. Type: ./configure –prefix= <xml-rpc-c installation path here, such as /usr/local>.
5. Type make.
6. Type make install.
7. Add the path of the xml-rpc-c installation + /bin to the PATH env var.
8. Add the path of the xml-rpc-c installation + /lib to the LD_RUN_PATH env variable.
9. Add the path of the xml-rpc-c installation + /lib to the LD_LIBRARY_PATH env var.

12. References

[1] Claire Tristram, “Handhelds of Tomorrow”, *Technology Review, 2002 at: http://www.techreview.com/articles/tristram0402.asp.*

[2] Tim Berners-Lee, James Hendler and Ora Lassila, “The Semantic Web”, *Scientific American, May 2001, at: http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html.*

[3] Marvin Minsky, “The Society of Mind”, *New York, Simon and Schuster, 1986.*

[4] Allen E. Milewski and Thomas M. Smith, “Providing Presence Cues to Telephone Users”, *Proceedings of CSCW, 2000.*

[5] A. Lee A. Girgensohn and K. Schlueter, “NYNEX Portholes: Initial User Reactions and Redesign Implications”, *Proceedings of the International ACM SIGGROUP on Supporting Group Work, ACM Press, 1997, 385-393.*

[6] E. S. Hudson and I. Smith, “Technique for Addressing Fundamental Privacy and Disruption Tradeoffs in Awareness Support Systems”, *Proceedings of CSCW, 1996.*

[7] W3C, “Web Services Activity, *at: http://www.w3.org/2002/ws/.*

[8] Sara Bly, Steve Harrison and Susan Irvin, “Media Spaces: Bringing People Together in a Video, Audio and Computing Environment”, *Communications of the ACM, 36(1), pp 28-47.*

[9] Paul Dourish and Sara Bly, “Portholes: Supporting Awareness in a Distributed Work Group”, *Proceedings of ACM CHI, 1992.*

[10] Adriana Vivacqua and Henry Lieberman, “Agents to Assist in Finding Help”, *Proceedings of ACM, CHI, 2000.*

[11] Eric Horvitz, Jack Breese, David Heckerman, David Hovel and Koos Rommelse, “The Lumière Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users”, *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, AAAI Press, 1998.*

[12] Kwan Hong Lee, "IMPROMPTU: Audio Applications for Mobile IP", *MIT Master's Thesis, Program in Media Arts and Sciences, 2001.*

[13] Henry Lieberman, "Letizia: An agent that assists web browsing", *Proceedings of the International Joint Conference on Artificial Intelligence, p.924-929, 1995.*

[14] Pattie Maes, "Agents that Reduce Work and Information Overload", *Communications of the ACM, Vol.37 No.3, July 1994.*

[15] Paul P. Maglio, Rob Barret, Christopher S. Campbell and Ted Selker, "SUITOR: An Attentive Information System", *Proceedings of ACM IUI, 2000.*

[16] Nitin Sawhney, Sean Wheeler and Chris Schmandt, "Aware Community Portals: Shared Information Appliances for Transitional Spaces", *Personal Technologies, 2000.*

[17] Ted Selker, "Cognitive Adaptive Computer Help (COACH)", *Proceeding of the International Conference on Artificial Intelligence, 1989.*

[18] Les Nelson, Sara Bly and Tomas Sokoler, "Quiet Calls: Talking Silently on Mobile Phones", *Proceeding of ACM CHI, 2001.*

[19] Matthew Marx and Chris Schmandt, "CLUES: Dynamic Personalized Message Filtering", *Proceedings of ACM CSCW, 1996.*

[20] John C. Tang, Nicole Yankelovich, James "Bo" Begole, Max Van Kleek, Francis Li and Janak Bhalodia, "ConNexus to Awarenex: Extending awareness to mobile users", *Proceedings of ACM CHI, 2001.*

[21] Lance Norskog et al., "SOX – Sound Exchange", *Version 10, at: http://www.spies.com/Sox/.*

[22] Eric Gamma, Richard Helm, Ralph Johnson and John Vlissides, "Design Patterns – Elements of Reusable Object-Oriented Software", *Reading, MA: Addison Wesley Longman, Inc., 1995.*

[23] Andrew Wood, "CAMEO: Supporting Observable APIs", *Proceedings of WWW5, Programming the Web Workshop, 1996.*

[24] A. Dix, J. Finlay, G. Abowd, R. Beale, "Human Computer Interaction", *Prentice Hall, 1993.*

[25] Apple Computer, Inc., "URL Access Manager", *Carbon 2002, http://developer.apple.com/techpubs/macosx/Carbon/networkcomm/URLAccessManager/urlaccessmanager.html.*

[26] H.J. Wang, B. Raman, C.-N. Chuah, R. Biswas, R. Gummadi, B. Hohlt, X. Hong, E. Kiciman, Z. Mao, J.S. Shih, L. Subramanian, B. Y. Zhao, A.D. Joseph, and R.H.Katz, "ICEBERG: An Internet-core Network Architecture for Integrated Communications", *Proceedings of IEEE Personal Communications (2000): Special Issue on IP-based Mobile Telecommunication Networks, Volume 7, 2000, Page 10-19.*

[27] M.L. Dertouzos, "The Oxygen Project: The Future of Computing", *Scientific American, 1999.*

[28] Daniel Salber, Anind K. Dey and Gregory D. Abowd, "The Context Toolkit: Aiding the Development of Context-Enabled Applications", *Proceedings of ACM CHI, 1999.*

[29] Anind K. Dey, Gregory D. Abowd and Andrew Wood, "CyberDesk: A Framework for Providing Self-Integrating Context-Aware Services", *Proceedings of the 3rd international conference on Intelligent user interfaces, January 1997.*

[30] J. Postel, "User Datagram Protocol (UDP)", *RFC768, 1980* at: http://www.ietf.org/rfc/rfc0768.txt?number=768.

[31] Alexander Guy, Carl Worth and Ken Causey, "The Familiar Project", *2001 at: http://familiar.handhelds.org.*

[32] Eric Kidd, "XML-RPC for C and C++" *at: http://xmlrpc-c.sourceforge.net/.*

[33] W3C, "Libwww - the W3C Protocol Library", *Version 5.3.2 at: http://www.w3.org/Library/.*

[34] Moez Mahfoudh, "ABYSS Web Server", *2000 at: http://abyss.sourceforge.net/.*

[35] Graham Cox, "MacZoop – The Framework for the Rest of Us", *Version 2.5.2, 2002 at: http://www.maczoop.com.*

[36] James Clark, “Expat – XML Parser Toolkit”, *Version 1.2, at: http://www.jclark.com/xml/expat.html.*

[37] BlueTooth. *http://www.bluetooth.com/dev/specifications.asp.*

[38] IEEE 802.11. *http://grouper.ieee.org/groups/802/11/index.html.*

[39] W3C, “Extensible Markup Language (XML)”, *Version 1.0, specification at: http://www.w3.org/TR/2000/REC-xml-20001006.*

[40] Userland, Inc. “XML-RPC”, *No version information, specification at: http://www.xmlrpc.com/spec.*

[41] W3C, “Simple Object Access Protocol (SOAP)”, *Version 1.1, specification at: http://www.w3.org/TR/SOAP/.*

[42] W3C, “Hypertext Transfer Protocol (HTTP)”, *Version 1.1, specification at: http://www.w3.org/Protocols/.*

[43] W3C, “Document Object Model (DOM)” *Level 1, specification, Version 1.0, at: http://www.w3.org/TR/REC-DOM-Level-1/.*

[44] Handspring, Inc. *http://www.handspring.com.*

[45] Apple Computer, Inc. “Applescript”. *http://www.applescript.com.*

[46] Power On Software, Inc. “Now Up-To-Date & Contact”. *http://www.poweronsoftware.com/products/nudc/.*

[47] ITU-T, “H.323”. *http://www.imtc.org/h323.htm.*

[48] Microsoft Corp, “Microsoft Entourage for the Macintosh”. *http://www.microsoft.com/mac/entouragex/.*