Non-Intrusive Computing

by

Hao Chen

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2008

© Hao Chen 2008

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Pervasive computing is an important trend today. It concerns devices and services in a smart space that interact with users in a simple, natural, and harmonious way. Many problems in this domain have been studied from different perspectives in various projects. However, one important characteristic of pervasive computing, which is how to make it non-intrusive so that users can focus on their tasks, has received little formal attention. Nowadays, many computing entities including smart devices, and software components, are involved in our daily lives, and users need to deal with them as well as with other people. Besides, people are easy to reach with multiple devices. We believe there should be a systematic way to help users avoid intrusive ones.

We propose a model for posing and answering two questions: will an interaction intrude on its receiver if delivered, and given that the interaction is deliverable, how can it be delivered effectively and not too overtly? With this model, the intrusion problem is analyzed and the essential factors are identified. A quantitative approach is used, so that factors have quantitative values for comparison and computation. We also apply context to refine them in order to achieve better results.

We then illustrate how to materialize the model and build a system whose design is inspired by the Jabber framework that includes a collection of standards, technologies, and projects for instant messaging. The discussion is at a general level that does not depend on Jabber. However, by choosing Jabber in implementation, we reuse existing software and technologies, and benefit from Jabber/XMPP standardization, its low entry barrier for application developers, and its rich community support.

The main contributions of our work are two-fold. First, we propose a model for intrusiveness in pervasive computing. Second, we address the problem at the system level by designing and realizing it. We also make use of standardized instant-messaging technologies, more precisely Jabber, in the system instantiation to reuse existing software, making the system more flexible and extensible.

Acknowledgements

I would like to thank my supervisor, Professor James P. Black, for his advice and suggestions, for his continuous support financially and on other aspects, for weekly conversations where he spent time and effort, even when he is away, for many tedious writing corrections, and for his patience and understanding through out my program. His sharp perspectives and wisdom taught me how to think and tackle problems and helped me come up with the ideas in this thesis.

I am also grateful to my committee members, Professors Johnny Wong, Ian McKillop, Paul A.S. Ward, Sidney Fels, and Catherine Burns for spending time in reading my thesis and providing me valuable advice.

My wife, Juan Wang, provides me family. Without her, I would not have been able to go through all of the difficulties I encountered. I thank her for all she is and all she has done for me.

I appreciate my colleagues and friends, Andrei Dragoi, Qiyan Li, Georgia Kastidou, Omar Khan, Herman Li, and Abdur-Rahman El-Sayed. I am also obliged to many other faculty members, staff, and students who helped my success.

Dedication

This is dedicated to my father, Haixiang Chen, and my mother, Morong Song.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Pervasive computing developed from distributed and mobile computing, and was first introduced by Mark Weiser of Xerox PARC in the early 1990s [55]. Ever since, it has received growing research interest. The overall goal of pervasive computing is to handle the complexity of many smart devices, software components, and human users, and integrate them together seamlessly. This involves multiple domains, from computing hardware, networks, systems, and user interfaces, to theory, modeling, algorithms, and so on. Projects in this area are as small as a few intelligent applications, or as big as umbrella projects with dozens of researchers. Weiser's visions of pervasive computing are explored in great breath and depth.

Among these visions, one is that users focus on their own tasks and let software and hardware components stay at the periphery [55]. This is also referred as calm technology [56]. In order to realize this vision, the system should help users deal with interactions.

Figure 1.1 depicts a scene in pervasive computing where there are two smart spaces, an office and a home, each of which consists of a number of users and devices. Spaces may intersect each other. Some devices are tightly associated with users, such as the notebook, the PDA (Personal Digital Assistant), the cell phone, the music player in the office, the smart watch, the tablet PC, and the cell phone at home, while others are more shared among users than associated with particular individuals. Examples are the printer, the whiteboard, the landline phone, the PC in the office, the smart fridge, the digital frame, the TV, and the X10 lamp (`http://www.x10.com`) at home. There could be other devices, represented by black dots (Figure 1.1). These devices have some computing and

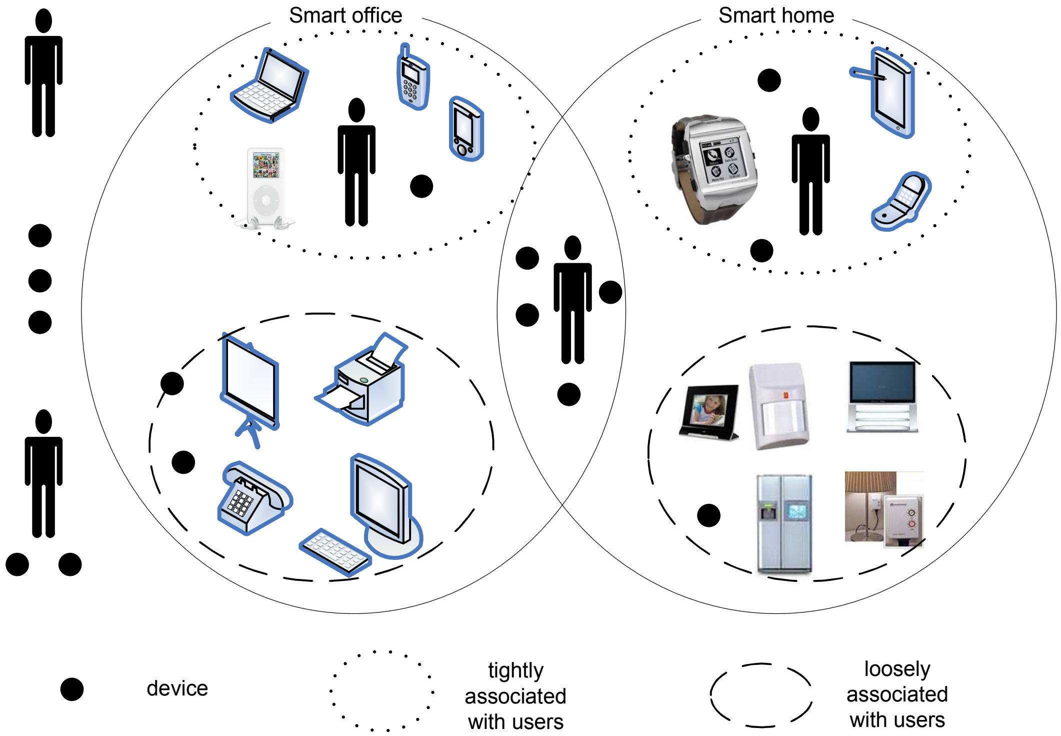


Figure 1.1: Overview of pervasive-computing environments

communication abilities, and can initiate, respond to, or deliver interactions. For example, the motion detector at home may send a warning message to the home owner once it detects an unexpected motion, or the whiteboard can display a picture for a user.

In such a strongly connected pervasive-computing environment, users face a large number of interactions. Some are desired while others are not, usually depending on how busy the users are and what those interactions are about. Besides, each interaction may involve different devices for delivery. For example, an interaction may be delivered to two devices, one of which is tightly associated with the receiver, and the other of which is loosely associated with the receiver. The reason for doing this is that the first type of device tends to have less computing power and smaller form factors, while the second type is usually larger and more advanced in these regards, hence more suitable for some interactions. However, this type of device may affect both the receiver and other people in the same environment, as is the case for the whiteboard, visible to everybody in the office. This may give rise to undesired situations, such as delivering a private high-resolution picture to the whiteboard during a meeting. All of these questions add more complexity in handling interactions in such environments.

We believe that users should be able to control these interactions, so that desired ones are delivered using appropriate devices and undesired ones are prevented from intruding on users. This is the main concern of this thesis. We use “non-intrusive computing” to refer to our work based on the following definitions of a few similar candidates.

> Disturbance - The interruption and breaking up of tranquility, peace, rest, or settled condition; agitation (physical, social, or political). [1]
>
> Interruption - A breaking in upon some action, process, or condition (esp. speech or discourse), so as to cause it (usually temporarily) to cease; hindrance of the course or continuance of something; a breach of continuity in time; a stoppage. [1]
>
> Intrusive - Of intruding character; characterized by coming or entering in an encroaching manner, or without invitation or welcome; done or carried out with intrusion. [1]

In the rest of this chapter, Section 1.1 first presents the motivation using a scenario, then we briefly introduce our approach to this problem in Section 1.2. The contributions of this work are discussed in Section 1.3, followed by the thesis organization in Section 1.4.

1.1 Motivating scenario

The focus of our work is non-intrusive computing that helps users control interactions. We use a home scenario to motivate our research and illustrate the problems that we are tackling.

Suppose Alice is working in her study at home. She is thinking about a net meeting that she will have with her supervisor and colleagues in about an hour. A chat window pops up on the monitor of her computer from her friend. Alice tells her friend that she is busy and will get back to her later. After reading some documents, Alice feels a bit hungry. Since there are still thirty minutes left before the meeting, she steps into the kitchen to have a sandwich. When she comes back to the study, she finds that she missed a call on her cell phone and a new pop-up window on the computer monitor. Both were from her supervisor saying that the meeting will start ten minutes early. She apologizes for missing the messages and starts to set up the net meeting.

After Alice finishes the meeting, she relaxes on a couch. At this time, the clothes dryer finishes. Since it is a smart dryer, it is configured to send a notification to the TV by displaying a line of text on the corner of the TV screen. Unfortunately, the TV is not turned on, so although Alice is right in front of the TV, she misses the message anyway, and before she remembers, the clothes are already wrinkled. After Alice folds them, two of her colleagues come to her place to discuss a project. During the discussion, Alice's son sends her a personal picture, and it is displayed on the digital frame next to Alice and her colleagues. Alice apologizes to her colleagues and turns off the digital frame to avoid other intrusions. Before the meeting is over, the cellphone of one of Alice's colleagues rings, and he apologizes for not setting it to vibrate, thus disturbing meeting participants.

This scenario shows that when a user is dealing with multiple interactions, the situation can easily get out of control. Some interactions are important and not intrusive, such as those from Alice's supervisor that Alice does want to receive, while others are indeed intrusive and Alice wants to avoid them, such as the one from her friend. Desired interactions may be missed if they are not delivered to the right devices. For example, the notification from Alice's supervisor should have been delivered to the phone in the kitchen, where she was at the moment. For another example, the message from the dryer should have been routed to a different device that was at least available. The scenario also depicts a scene with multiple people in the same location, where uncontrolled interactions cause intrusions to other people if they are not delivered to the right devices using correct notifications, such as the private picture displaying on the digital frame and the cellphone ringing during the meeting. It would have been better if the picture was displayed on Alice's laptop, which is only viewable to herself, and the cell phone vibrated instead of rang.

These situations can be exaggerated in a pervasive- and ubiquitous-computing environment that is usually shared by people and saturated by entities and interactions. Besides, users are tending to acquire more smart handheld devices, which in turn makes them more open and vulnerable to interactions. If users receive no support in managing these interactions and devices, they may be frustrated easily and thus unable to focus on their own tasks.

1.2 Our approach

Given the scenario in Section 1.1, it is necessary to build a system that selects interactions and delivery devices. It is also necessary to find a model that can help us understand and

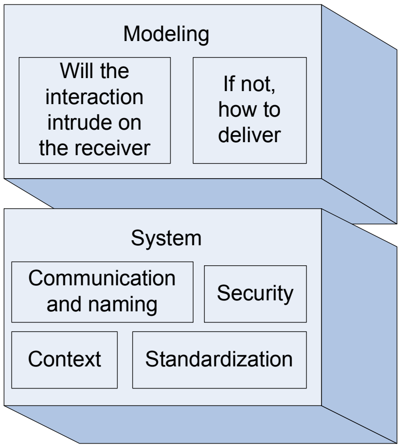


Figure 1.2: Two-tier approach

generalize these problems and also guide system construction. Figure 1.2 is our two-level approach to non-intrusive computing.

At the upper level, there is a model that abstracts interactions and deliveries, where key elements are discussed. The model focuses on two particular questions. The first is how to determine whether an interaction will intrude on the receiver if delivered. The second is that, given a non-intrusive interaction, how to deliver it properly. Chapter 2 contains a detailed discussion of the model.

Then at the lower level, we use the model to guide the construction of a non-intrusive-computing system in Chapter 5. Aside from the functionalities implied by the model, the system follows a few principles.

- It should provide communication support for all entities in a pervasive-computing environment, because the environment involves many devices and software components, and communication happens more often than ever.
- It should provide naming support in this environment, to identify human users and software and hardware entities consistently.
- It needs to manage context, and enable entities to access it in a flexible way.
- It should provide security support.
- It should leverage standardized technologies and existing components for extensibility and adoption.

Our system is inspired by and built on Jabber [28], a collection of standards, technologies, and software projects for instant messaging. Although our work is aimed at intrusiveness in pervasive computing, our system has elements and qualities of pervasive-computing middleware. It distinguishes itself from previous pervasive-computing middleware and platforms because of Jabber's standardization, architecture, extensibility, and other aspects.

1.3 Thesis contributions

The goal of our work is to help users deal with interactions, so that they receive non-intrusive ones properly. This will allow them to focus better on their current tasks without being disturbed by intrusive interactions. This area has received little attention, and our work fills a void and materializes one of the Weiser's visions of pervasive and ubiquitous computing.

There are two major contributions.

1. The non-intrusive-computing model captures key elements of the intrusion problem in pervasive-computing environments. The model is separated into two stages, the filter and delivery stages. They deal with different questions, and are related naturally. The model is concise and comprehensive. It not only considers the intrusiveness to the receiver, but also to other users in the same space. The model is similar to the administrative-assistant or secretary models in real life, in which a secretary decides whether a message or a phone call is important enough to interrupt her supervisor and whether she should forward the message to his cell phone or knock on the door.

2. We develop such a system for non-intrusive computing, based on Jabber technologies. The system uses standardized technologies. It is practical and extensible, and complies with the model. Although Jabber provides many features and facilities (see Section 5.2), it does not solve the system problem directly. We develop various components, such as a context manager and the model implementation, and integrate them with Jabber.

We also make a number of other contributions. We apply context in a pluggable way. More or better context can be injected into the system to refine the model and help the system make more appropriate decisions. We also use a quantitative approach that is able to model elements and perform comparisons and computations. The target devices for delivery are also treated quantitatively. We identify two types of delivery candidates and their characteristics with respect to effectiveness and overtness: a private delivery candidate has negligible overtness and a public delivery candidate has the same effectiveness as the overtness. The candidate-selection process is a knapsack problem, as it requires both low overtness and high effectiveness. Although this problem is NP-complete, categorizing candidates according to public and private delivery helps simplify the problem in some cases, so that the results may be found in linear time.

1.4 Organization of the thesis

This dissertation is divided into six chapters, the introduction in Chapter 1, the modeling and performance analysis in Chapters 2 and 3, the previous work on systems and our design and prototyping in Chapters 4 and 5, and the conclusions in Chapter 6. As we can see, the flow generally follows the three major contributions in Section 1.3.

In Chapter 2, non-intrusive computing is analyzed and modeled in detail. We first review previous work on interruptions and intrusiveness, then model the problem domain with two stages, the filter and the delivery stages, with detailed discussions of each of them. A user study is also conducted to investigate how users understand the model and its parameters.

Chapter 3 analyzes the performance of the model theoretically, in conjunction with a few experiments to confirm the analysis. We measure the filter stage only, because the delivery stage needs much configuration and results are less general, so it will not provide any insight into the model or the system.

In Chapter 4, we start to describe the system instead of the model. In particular, Chapter 4 surveys previous pervasive- and ubiquitous-computing systems. There are large number of such systems, and so only representative ones are selected and presented. Chapter 5 describes a non-intrusive-computing system, including its design, enabling technologies, overhead evaluation, and a case study of how such a system could be used.

Lastly, Chapter 6 concludes the thesis and highlights future research in this area.

Chapter 2

Modeling non-intrusive computing

In this chapter, we discuss modeling non-intrusive computing [13]. Before describing the model, we review related research on intrusiveness in Section 2.1. Section 2.2 introduces the overall model. Sections 2.3 and 2.4 continue the discussion of the stages of the model. Section 2.5 describes a user study to validate the model. Section 2.6 summarizes this chapter.

2.1 Previous work on intrusiveness

From a distributed-system point of view, much previous work tends to design the pervasive-computing environments to cooperate with users in an automatic fashion, so that users can concentrate on their tasks, instead of being disturbed because of the complexity of the environment. An example is the Aura project [19], which describes tasks in an environment-independent way, so that tasks can move automatically along with users from one environment to another. Another example is the Gaia project [43], which develops a context file system where relevant data can be mounted according to the current context of the users. These systems focus on integrating components and reducing users' interventions in a pervasive-computing environment, but they do not address the intrusiveness problem of interactions directly. More detailed descriptions of Aura and Gaia are in Chapter 4, together with other systems.

From human-computer-interaction point of view, interruptions caused by computing software and devices have been investigated in many ways. The common goals are to study what effects an interruption may have on users and their tasks, and to develop models and systems to manage interruptions. Adamczyk *et al.* [3] discuss interruptions at different stages of a task, and try to find proper interruption moments, even defer interruptions if necessary, to mitigate the disruptive effects on the user's current task. Then they use psychological theory to model tasks in order to identify opportune moments and achieve finer-grained temporal reasoning [4]. Bailey *et al.* [7] study how much task performance is affected by interruptions from computer applications. The results show that users perform worse when there are interruptions than when there are not. Andrews [5] describes a unified interruption-management approach that combines tools, social practices, and policies to help users control their attention and improve productivity. QnA [6] is an augmented instant-messaging client that balances user responsiveness and performance. It first analyses an instant message and classifies it. For example, it tries to determine whether a message contains a question. If it does, the QnA client indicates to the receiver that the sender is expecting a response; otherwise, it does nothing. This helps the user decide whether to respond to the message or to stay on the current task.

Much interruption research has also been applied to cell-phone use while driving. In these cases, if someone calls the driver's cell phone, it will cause distraction, which could be risky. People are relying more and more on cell phones today. In the world, cell phones are in use by 32.5% of the population in 2005 [58]. Evidence shows [24] that using a cell phone while driving impairs driving performance, and this impairment may lead directly to accidents. According to the U.S. Department of Transportation, inattentive driving accounted for 6.4% of crash fatalities in 2003 [27].

Using a cell phone requires both physical involvement, such as dialing and viewing, and mental involvement, such as discussion. Both distract from driving. Hands-free phones, which help users reduce physical involvement, do not necessarily improve safety [42]. Approximately 50 countries have passed legislations to ban cell phones while driving [64].

The problem of cell phones distracting drivers may be alleviated in two ways. One approach is to improve the user interface to reducing eye viewing [20, 34]. The other, more relevant to our research, is to provide remote traffic context to callers [23, 47]. If the caller is made aware of road context, he can moderate the conversation, such as stopping talking. The callee, who is the driver, can concentrate more on the road traffic. Although we are not focused on vehicular research, our system fits well to this situation, as it uses the importance of an interaction, determined by the caller, and the willingness of the callee to decide whether this interaction causes any intrusiveness if delivered, as discussed in Section 2.3.

As we can see, research on interruption usually has a notion of tasks and their models and focuses on measuring the negative effects on tasks caused by interruptions. Our work models interactions themselves directly.

A more relevant work is an agent-based approach to minimizing intrusiveness in a meeting environment [41]. Its main idea is the following. In a meeting scenario, participants have their own devices, such as a personal laptop; they also share some public devices, such as a whiteboard in the meeting room. Displaying a message on a public device is more intrusive than displaying it on a private device, as all participants can see the whiteboard but not the screen of a laptop. Each user has an agent maintaining the user's interests and making decisions for him. When a message arrives for a user, the agent checks with other agents to see if they are interested in this message. If so, the message will be displayed on a public device; otherwise, it goes to a private device. The limitation of this work is that it only applies to a situation where multiple users have shared devices, for example a meeting scenario. Our work, instead, is aimed at a more general question: given an interaction, is it going to intrude on the receiver?

Existing systems such as electronic mail, telephone, and instant messaging (IM) are also relevant to the intrusion problem in general.

Electronic mail and telephony rely on distributed communication systems, which greatly facilitate human interactions. Telephones are usually intrusive, because people do not know when the phone will ring, who it is from, or what it is about. Some technologies can alleviate the problem, such as call forwarding to voice mail, call blocking, or caller identification. But they are insufficient and inflexible. Call forwarding redirects all of the calls, even important ones that do need direct conversation with the user. Call blocking only works for the numbers that are blocked, but calls from others can still be intrusive and disturbing. Displaying caller ID does not help much when the ID is not recognizable, such as wrong numbers. Conventional telephones notify users only by ringing. Cell phones have more choices, such as vibrating, visual blinking, or even remaining completely silent. These features are more useful when cell-phone users are in a public environment, such as a meeting, a class, and so on, to make them less intrusive to other people. However, if a phone call is not what a user wants at the moment, it is still intrusive, no matter how it notifies the user.

An electronic-mail system tends to be less intrusive, as receivers decide when to read messages, which is essentially due to the time-decoupling of the sender and the receiver. Although many electronic mail clients can check new messages at a pre-set frequency and notify people, it is still time-decoupled because transferring a message from a sender to a receiver might take a long time, and receivers might not read messages immediately upon receipt.

IM is becoming more and more popular. Instances are MSN messenger, AOL messenger, Yahoo messenger, ICQ, Google Talk, and so on. In an IM system, users can set *presence* describing their availability status, such as "do not disturb," and the presence is delivered to users who subscribe to it. Because of the presence, the message sender knows if the receiver is available and can decide to send or not. This reduces the intrusiveness if senders respect the presence. But, if the sender sends a message, it will be delivered to the receiver, regardless of her presence.

Previous work shows that this intrusion problem is important and deserves much research. Existing systems deal with the problem in different ways but with various limitations. Next, we abstract our research problem with an overall model.

2.2 Non-intrusive computing: Overall model

In general, interactions can be labeled as time- or reference-coupled or decoupled [10]. Time-coupled interactions are synchronous; reference-coupled ones have specific responders. We are only interested in time- and reference-coupled interactions. Those that are time-decoupled or reference-decoupled do not pose similar intrusion problems. We also focus on interactions that involve only one interaction initiator (sender) and one responder (receiver) as this is the most common and basic case for interactions. One-to-multiple interactions are considered future work.

An interaction process extends from when a sender initiates the interaction until it is delivered to some device and a receiver is notified in some way. For example, a sender initiates a chat with one of his friends, and the interaction is delivered to the receiver's cell phone, which rings softly to inform the receiver.

This process can be divided naturally into two stages. The first determines whether the interaction is deliverable to the receiver. If it is not, it should be prevented, since it will intrude upon him/her. The second stage, given the interaction is deliverable, determines

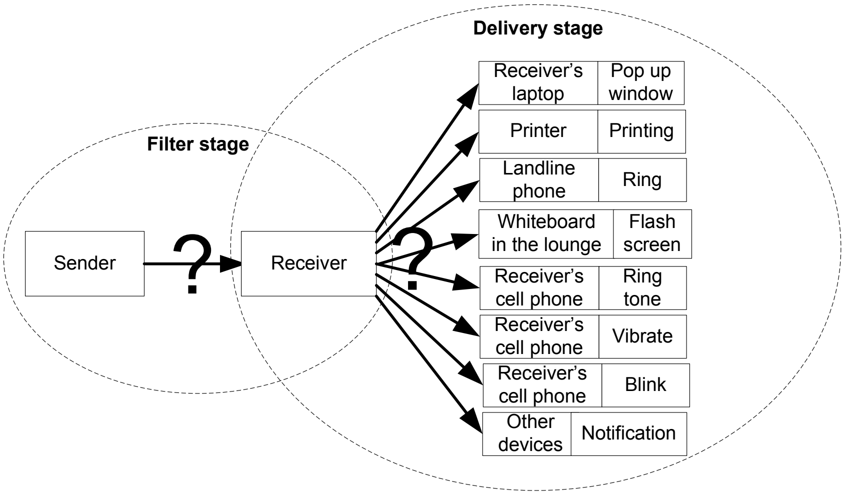


Figure 2.1: Overall modeling

how to deliver it to the right device and notify the receiver appropriately. We call the first part the filter stage, and the second the delivery stage, as illustrated in Figure 2.1.

The filter stage only involves a sender and a receiver, as we can see in Figure 2.1. The interaction can be of any form. For example, it can be an instant message, a file transfer, a remote procedure call (RPC), a multimedia conversation, and so on. In order to determine the deliverability of an interaction, we need to know the importance of the interaction and the willingness of the receiver. The determination procedure depends on the presence model of instant-messaging (IM) systems (see Section 5.2.4). A detailed description on how to determine deliverability is in Section 2.3.

Once we decide that the interaction is deliverable, hence not intrusive to the receiver, the interaction takes place and arrives at the receiver side. This is the delivery stage in Figure 2.1. We can see that on the receiver's side, there could be multiple devices available at the moment, such as a laptop, a cell phone, or a whiteboard. Each of them may have one or more mechanisms for notifying the receiver. For example, a cell phone can ring, vibrate, or blink. In order to choose a proper device and a notification method, we should consider two factors.

First, the interaction should be delivered to the receiver in such a way that it is rare for the user to miss it. We call this effective delivery. For example, if a user is away from her office, it is better to deliver interactions to his cell phone than to his desktop computer. For another example, if a user is listening to some music when an interaction arrives, she should be notified with a pop-up window instead of beeping, because she might not hear it. The fact that we are dealing with time-coupled interactions results in the necessity of effective delivery.

Second, a user might be in a shared environment with other people. Choosing an inappropriate device or notification method may cause intrusion on other users nearby. For example, during a meeting, if a private interaction for a user is delivered to a whiteboard or to a phone that rings, other users either see it on the whiteboard or hear the ringing. We call this the overtness of the delivery. We want the delivery to minimize overtness.

Combining these, interactions should be delivered to proper destination devices in proper ways, so that receivers notice them and surrounding people are not disturbed. In the next two sections, we discuss the details of our work in the two stages. In each stage, we present and analyze factors that affect intrusion. We use a quantitative approach in dealing with the problems.

2.3 Non-intrusive computing: Filter stage

In order to describe the model and its parameters clearly, we use A and B to represent the sender and the receiver of the interaction, $A \rightarrow B$. The same notation is used elsewhere in this thesis.

Intuitively, if B does not want to interact, for example because of being busy, delivering an interaction from A to B tends to intrude on B. We use *willingness* to describe this factor. However, on the other hand, if the interaction is important enough, B will want to accept it. So the interaction is still deliverable and does not cause intrusion to B. We use *importance* to denote this factor. The willingness and the importance can be used to decide whether an interaction is deliverable or not. A non-deliverable interaction is intrusive to the receiver if delivered; a deliverable one does not cause such intrusion. The approach we take is to quantify these two factors and then present a comparison model to deduce a value of deliverability. The model, although simple, can produce acceptable results. To validate this point, we also describe a more complicated model using fuzzy logic, and show that these two models generate similar results.

We next quantify these two factors and present the two models. However, there may be other factors that affect both the willingness and the importance, such as the role relationship between the sender and the receiver, the receiver's willingness preferences in particular situations, and so on. Hence, how to modify the importance and willingness and a refined model are described from Sections 2.3.5 to 2.3.8.

2.3.1 Willingness

To determine if an interaction is intrusive to a receiver, B, we need to know how willing B is to interact. The willingness changes dynamically, and is entirely at the discretion of B. We use W_B to denote B's willingness. The quantization is the following.

The lower bound of the willingness should indicate that the entity is completely unwilling to interact, and the upper bound should indicate that the entity is completely willing. Thus, we need similar values for both "willingness" and "unwillingness." These values should be finite. It is also necessary to perform various comparisons and transformations on these finite ranges of values, from completely unwilling to completely willing. Thus, we choose $[0,1]$, which has appealing intuitive and mathematical properties, such that 0 means completely unwilling, 1 means completely willing, and unwillingness is equal to 1 minus willingness. Besides, we can think of the unwillingness as a threshold for importance so that only those interactions whose importance is greater than the importance threshold should be delivered. We use unwillingness and importance threshold interchangeably.

Although a willingness of 0 is acceptable, it does not have any practical value, because 0 would mean the current entity does not want to interact at all, and the interaction should not, or even cannot, take place, such as when an IM user is off-line, and so there is no intrusion possible.

Note that social-networking protocols usually describe a user with some status. For example, in Skype (`http://www.skype.com`), a user can be "skype me," "away," "not available," "do not disturb," and so on. These statuses can be mapped to different values of willingness, or used to infer the willingness together with other information (see Section 5.3.2).

2.3.2 Importance

The importance of an interaction also affects the intrusiveness. If an interaction is of little importance, it tends to be intrusive, such as chatting. If an interaction is important, such as informing a user of an accident, then it should perhaps override the unwillingness, even if the receiver does not wish to interact. For example, Bob is in a meeting, with little willingness to interact. Bob receives a message that his mother was sent to the hospital due to health reasons. Clearly, Bob will not think this interaction is intrusive to him, rather, he will be grateful to the person who informs him.

We mention above that the willingness is determined by the receiver of an interaction. In contrast, the importance of an interaction is determined by the sender, because we believe the sender must know the purpose of the interaction, which makes him or her capable of determining the importance. We use I_A to represent the importance of an interaction initiated by a sender A.

The lower bound of I_A indicates the interaction is meaningless, and the upper bound indicates that the interaction is of the utmost importance, and should occur immediately. As above, we choose 0 for the lower bound and 1 as the upper bound for importance, so it is comparable with the willingness. Again, we include 0 for modeling purposes, although it represents meaningless interactions.

2.3.3 The comparison model

Obviously, the importance and unwillingness are competing factors that affect the deliverability of an interaction. The higher the importance, the higher the deliverability; the higher the unwillingness, the lower the deliverability. Given an interaction $A \rightarrow B$, the importance I_A and the willingness W_B, the comparison model for determining the deliverability is shown in Figure 2.2.

The model compares the importance I_A against the importance threshold, that is, $1 - W_B$. The difference is the deliverability, d. If I_A is greater than or equal to $1 - W_B$, the interaction is deliverable, otherwise it is not. There is a boundary condition where $I_A = 1 - W_B$, in which case, we assume that this interaction is not intrusive, because the purpose of this model is to reduce the intrusiveness of interactions, not to block them.

The relationships among deliverability, importance, and unwillingness are given in Figure 2.3. The deliverability increases with importance and decreases with unwillingness.

The model, being simple and straightforward, produces acceptable results. More complex models, such as those using fuzzy logic [61], generate similar, if not the same,

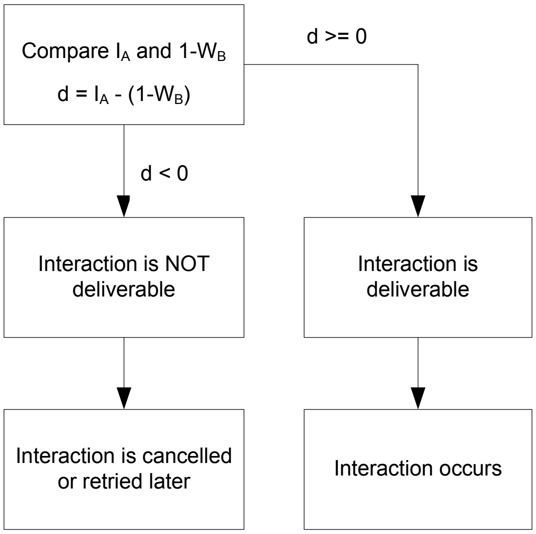

Compare I_A and 1-W_B
d = I_A - (1-W_B)
d >= 0
d < 0
Interaction is NOT deliverable
Interaction is deliverable
Interaction is cancelled or retried later
Interaction occurs

Figure 2.2: The comparison model

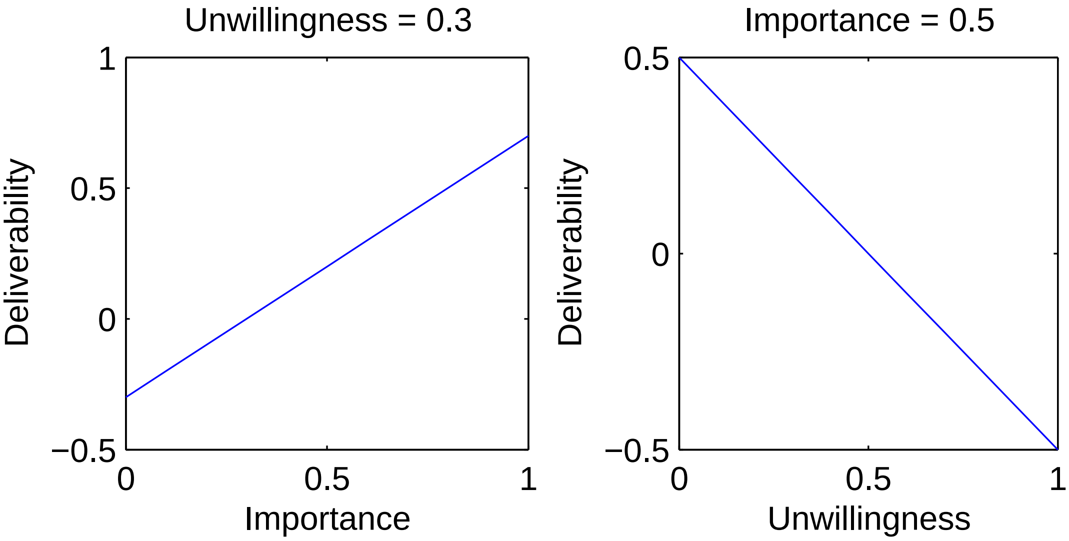


Figure 2.3: Relationships among importance, willingness, and deliverability, using the comparison model. Left: unwillingness = 0.3, right: unwillingness= 0.5

results as our comparison-based one. In the next section, we discuss an approach based on fuzzy logic. Note that it only validates our model. In practice, we will not adopt it because, compared to our model, it is more complex, requires more computing, and makes the same decision on whether to deliver the interaction. Later, we will use context to improve the model in Section 2.3.8.

2.3.4 The fuzzy-logic model

Fuzzy logic [61] was invented by Lotfi A. Zadeh at the University of California, Berkeley in 1965. It is derived from traditional logic, where a predicate can only be true or false. In other words, traditional logic maps a subject to a set with only two values, 0 and 1. For example, suppose Alice is female and Bob is male, we have $male(Alice) = 0$, $male(Bob) = 1$. *Male* is a crisp set. However, for some predicates, it is not obvious whether they are absolutely true or false. Instead we can say that they are true with a certain degree of truth. For example, if Alice and Bob are 15 and 30 years old respectively, we can say that Alice is quite young, while Bob is still young but not that young, depending on how we define young. So if we formalize the definition for young (see below), then we have $young(Alice) = 1$, $young(Bob) = 0.5$. In this case, *young* is a fuzzy set that maps a subject (a person) to a value between 0 and 1 to indicate the degree with which the subject belongs to the set. The definition of *young* is called the membership function. Fuzzy logic is based on fuzzy-set theory plus a number of inference rules that we describe next. For example, we might define $young(x)$ as:

$$young(x) = \begin{cases} 1 & x \leq 20 \\ -\frac{1}{20}x + 2 & 20 < x < 40 \\ 0 & x \geq 40 \end{cases}$$

In our case, given a value of importance or unwillingness, it is more natural to say it is high or low to a certain degree than to claim it is high or low in an absolute sense. Thus it may be suitable to apply fuzzy-set theory and fuzzy-logic control to our scenario.

We define three fuzzy sets for importance, *high*, *low*, and *zero*. Each of these maps an importance value to a membership value describing the degree of the importance being in the set. The importance I has range $[0,1]$, hence the universe of discourse has range $[0,1]$. The definitions of the three sets are the following.

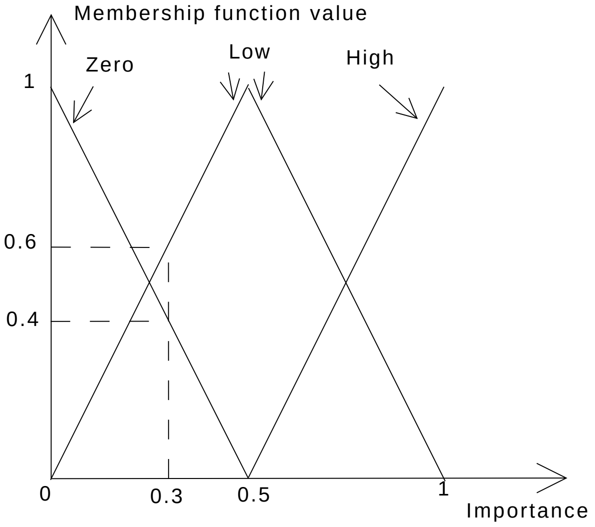


Figure 2.4: Fuzzy sets for importance

$$
\begin{aligned}
zero &= \begin{cases} -2I+1 & 0 \leq I < 0.5 \\ 0 & 0.5 \leq I \leq 1 \end{cases} \\
low &= \begin{cases} 2I & 0 \leq I < 0.5 \\ -2I+2 & 0 \leq I \leq 1 \end{cases} \\
high &= \begin{cases} 0 & 0 \leq I \leq 0.5 \\ 2I-1 & 0.5 < I \leq 1 \end{cases}
\end{aligned}
$$

The graphical representations of these functions are in Figure 2.4. As we can see, an importance value of 0.3 belongs to fuzzy set *zero* with truth degree 0.4, but to fuzzy set *low* with truth degree of 0.6. We choose the same fuzzy sets and membership functions for the unwillingness.

We also define fuzzy sets for deliverability in a similar way in order to apply fuzzy rules and perform the delivery computation later. Note that the universe of discourse for deliverability, d, is $[-1, 1]$. The fuzzy-set definitions for deliverability are illustrated in Figure 2.5.

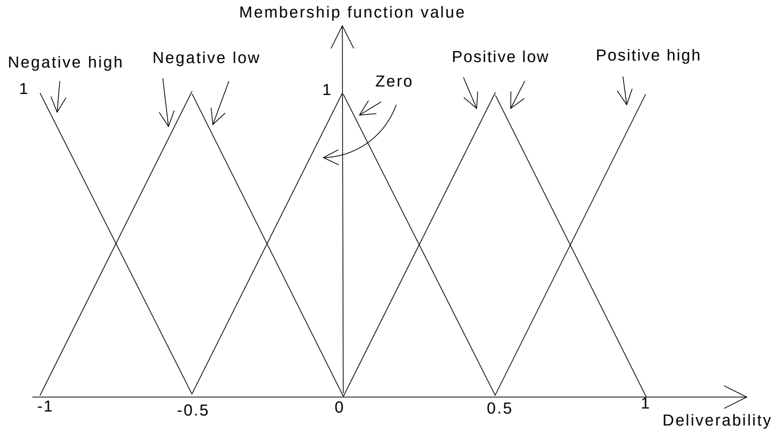


Figure 2.5: Fuzzy sets for deliverability

$$
\begin{aligned}
\textit{negative high} &= \begin{cases} -2d-1 & -1 \le d < -0.5 \\ 0 & -0.5 \le d \le 1 \end{cases} \\
\textit{negative low} &= \begin{cases} 2d+2 & -1 \le d \le -0.5 \\ -2d & -0.5 \le d < 0 \\ 0 & 0 \le d \le 1 \end{cases} \\
\textit{zero} &= \begin{cases} 0 & -1 \le d \le -0.5 \\ 2d+1 & -0.5 < d \le 0 \\ -2d+1 & 0 < d < 0.5 \\ 0 & 0.5 \le d \le 1 \end{cases} \\
\textit{positive low} &= \begin{cases} 0 & -1 \le d \le 0 \\ 2d & 0 < d \le 0.5 \\ -2d+2 & 0.5 < d \le 1 \end{cases} \\
\textit{positive high} &= \begin{cases} 0 & -1 \le d \le 0.5 \\ 2d-1 & 0.5 < d \le 1 \end{cases}
\end{aligned}
$$

The goal is to use an importance and an unwillingness to determine the deliverability. Now we show how to use fuzzy logic to achieve this. The fuzzy-logic control consists of a number of rules, expressed in terms of fuzzy sets. Table 2.1 describes rules used in this scenario. There are nine rules for different values of importance and unwillingness. For example, when the importance is *low* and the unwillingness is *zero*, the deliverability

Fuzzy-control rules for deliverability	Unwillingness *zero*	Unwillingness *low*	Unwillingness *high*
Importance *zero*	*zero*	*negative low*	*negative high*
Importance *low*	*positive low*	*zero*	*negative low*
Importance *high*	*positive high*	*positive low*	*zero*

Table 2.1: Fuzzy-control rules

should use fuzzy set *positive low*. When the importance is *zero* and the unwillingness is *high*, the deliverability is *negative high*.

Generally, these rules are intuitive and straightforward. Each one has two inputs, the importance and the unwillingness, and one output, the deliverability, expressed in terms of fuzzy sets. When the given importance and unwillingness belong to these sets according to the set definition, then the rule is applied by evaluating the membership values of the input, i.e., the importance and the unwillingness. The larger membership value is discarded while the smaller one is transferred to the output side by cutting the whole area defined by the corresponding fuzzy set for the deliverability at the height of that value. This process is shown in Figure 2.6.

Suppose an importance I is 0.3, and an unwillingness Unw is 0.6. By definition, we know this importance belongs to fuzzy sets *zero* and *low* with degrees 0.4 and 0.6 respectively, and the unwillingness belongs to *low* and *high* with degrees 0.8 and 0.2 respectively. Thus there are four rules to be applied. Figure 2.6 only shows two of them. Steps 1 to 3 correspond to "if the importance is *zero*, and the unwillingness is *low*, then the deliverability is *negative low*." Steps 1 and 2 evaluate the degrees of variables belonging to fuzzy sets. The lower degree value is 0.4, thus we end up with the shaded area in step 3. Step 4 to 6 illustrate the rule that if the importance is *zero*, and the unwillingness is *high*, then the deliverability is *negative high*. Thus, we have another shaded area together with the previous one in step 6. We repeat the same process for all applicable rules and merge all areas into a combined one, indicated by thick lines at step 7. Note that a few steps corresponding to other applicable rules are omitted between 6 and 7, and that there are two rules whose outputs are *negative low* for the deliverability. The X coordinate of the centroid of this area is the deliverability that we desire, which is -0.22 in this case. This procedure is also known as defuzzification. Specifically, this technique is MAX-MIN inference [48], where we choose the minimum of the degrees of input parameters, and maximize the areas generated by each rule. There

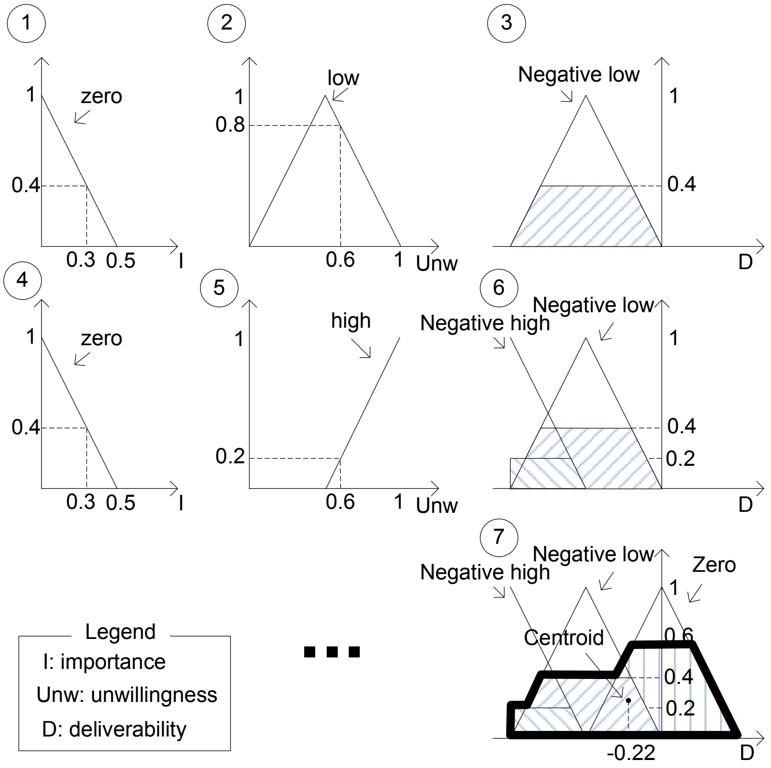


Figure 2.6: Fuzzy-logic computing

are other techniques with which we could experiment. However we choose the min-max one because of its popularity. Algorithm 2.1 summarizes the computation.

We now show the results that we get from this fuzzy-logic model in Figures 2.7 to 2.9. From these figures, we can see that when the importance and unwillingness are far apart, the deliverability results are noticeably different, while when the importance and unwillingness values are close, the results from the two models are also close. Although these two models produce different results, they are the same in terms of filtering intrusive interactions. The reason is the following. Since we are dealing with time-coupled interactions, for any such given interaction, there are only two options, delivering or not delivering. Furthermore, if the deliverability is greater than zero, we deliver the interaction; otherwise, we do not. Clearly, both models are consistent when it comes to the polarity of the deliverability. This being said, although the fuzzy-logic model fits well into this scenario, it acts identically to the comparison model of Section 2.3.3. Thus, we will not use it in our implementation and the purpose of discussing it is to validate our comparison model.

We know that the sender decides the importance based on the purpose of the inter-

Algorithm 2.1 Using fuzzy logic to compute deliverability

```
/*
 * given an importance ($imp$) and unwillingness ($unw$)
 * this algorithm outputs the deliverability using
 * fuzzy logic
 *
 * $n$: number of rules
 * $rule[i]$: the $i$th rule, $i$: $0 \sim n-1$
 * $zero(imp)$, $low(imp)$, and $high(imp)$ are the truth degree
 * of $imp$ being $zero$, $low$ and $high$. The same for $unw$.
 */
initialize $n$ and $rule$;
compute $zero(imp)$, $low(imp)$, and $high(imp)$;
compute $zero(unw)$, $low(unw)$, and $high(unw)$;
$i \leftarrow 0$;
$area \leftarrow 0$;
while $i < n$
  if $rule[i]$ can be applied, i.e., $imp$ and $unw$ belong to
  fuzzy sets specified in $rule[i]$
    $height \leftarrow min(fuzzy\_set(imp), fuzzy\_set(unw))$;
    $area \leftarrow area \cup$ corresponding fuzzy set for
             deliverability under $height$;
  $i \leftarrow i + 1$;
compute the centroid of $area$;
return the x coordinate of the centroid;
```

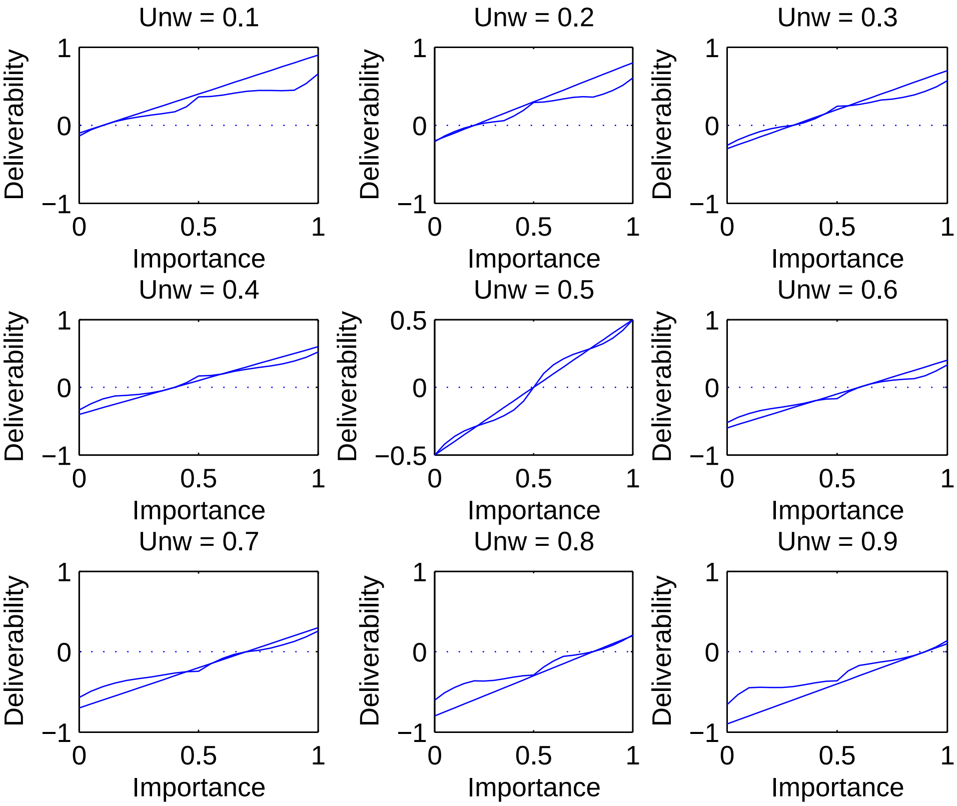


Figure 2.7: Deliverability. Curved lines: from the fuzzy-logic model; straight lines: from the comparison model; Unw: unwillingness

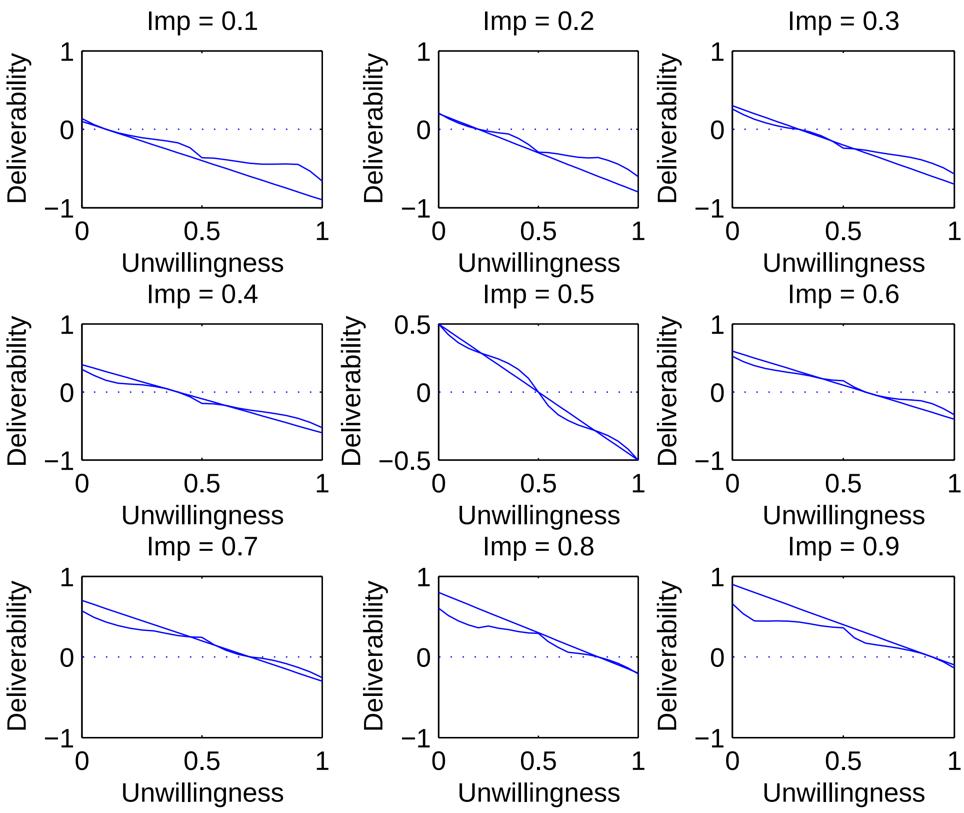


Figure 2.8: Deliverability. Curved lines: from the fuzzy-logic model; straight lines: from the comparison model; Imp: importance

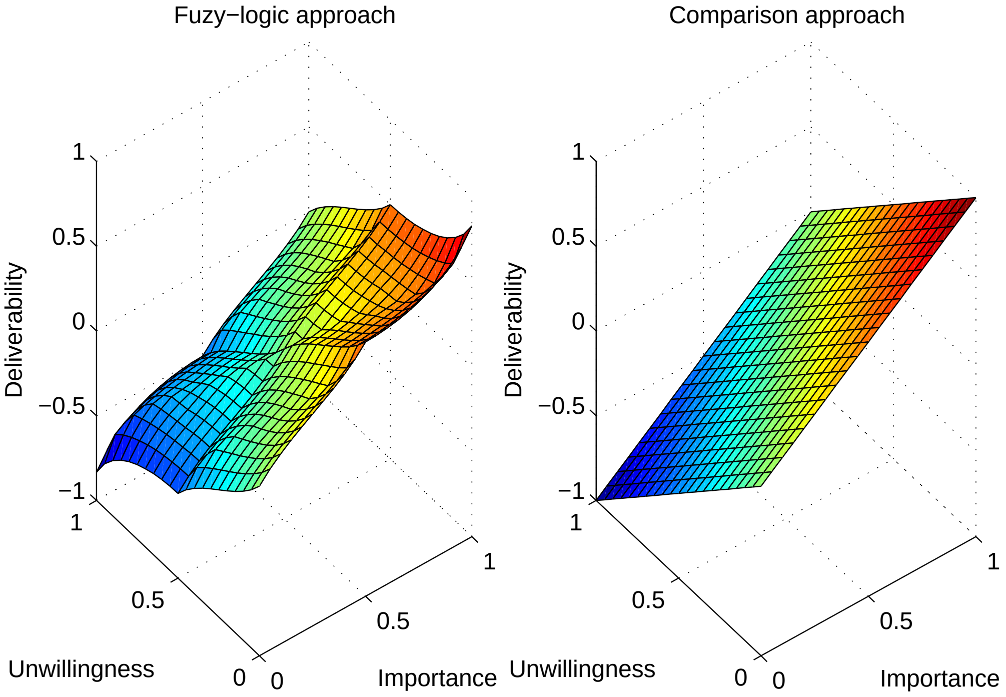


Figure 2.9: 3D deliverability. Left: deliverability computed with fuzzy logic; right: deliverability computed with comparison

action, and the receiver sets the willingness. However, this is not precise enough for a number of reasons. For example, because importance is relative, an interaction may be important to the sender, but less important to the receiver, or vice versa. When this kind of situation occurs, receivers will be annoyed, either by receiving non-deliverable interactions or by not receiving deliverable ones. We need to take this into consideration. As another example, the willingness describes a general situation, but there are always exceptions. A person may have a low willingness, as he is working on something. So this person is not interested in interactions in general, but only in related ones. Furthermore, since senders and receivers are dealing with many interactions, the importance and the willingness should be set as automatically as possible. Next, we discuss using context to address these problems.

2.3.5 Modifying importance and willingness

The comparison model in Section 2.3.3 depends on accurate importance and willingness. Users may set them. However, because of the dynamics of pervasive-computing environments, it is impractical to always rely on users to specify and update values. Hence, the system needs to modify users' input or infer new importance or willingness using context.

Context can be applied in different ways. In the left half of Figure 2.10, multiple pieces of context are processed by a complex modifier simultaneously. This modifier acts as a black-box algorithm. On the contrary, the right half of the figure shows a pipeline of context processing. Each "atomic" modifier only takes one piece of context and outputs a modified value of the importance or willingness. The advantage of this approach is that it is easy to expand and add more elements of context. The disadvantage is that different context elements may be correlated, which makes the black-box approach more appropriate. Next, we present some examples of these modifiers in Sections 2.3.6 and 2.3.7.

2.3.6 Atomic modifiers

Atomic modifiers can be applied for both importance and willingness. They change the original I_A or W_B based on one piece of context, so that the new value is more appropriate or accurate than before. In this section, we describe two modifiers. One is for importance

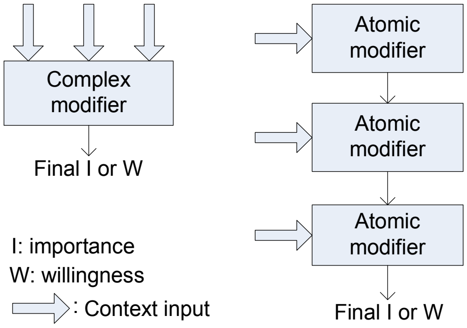


Figure 2.10: Applying context in the filter stage. Left: black-box approach; right: pipeline approach

based on the role relationship between the sender and the receiver. The other is for the willingness based on the receiver's interest in particular topics.

As we briefly mention at the end of Section 2.3.4, the importance is relative to the entities that participate in the interaction. More precisely, it can be relative to the roles of these entities. Consider a failed student who tries to interact with an instructor, where the interaction is important to the student but perhaps not to the instructor. In this case, the comparison needs to reduce the importance to prevent the interaction from happening when the instructor is not very interested in interacting. Or suppose a manager is trying to send out a meeting notice to his employees. A meeting notice probably is not the most important interaction among all kinds of interactions in a company environment. So if an employee is busy, she can present a low willingness, to avoid intrusions. However, this interaction is from the manager, so employees will not want to miss it. The comparison model should let the interaction happen by increasing the importance.

In the first example above, the role of the initiator is lower than the receiver's, and we need to reduce the importance, while it is the opposite case in the second example. Obviously, the modifier needs to increase or decrease the importance according to the relative roles of the senders and receivers.

Assume roles are organized into a tree as in Figure 2.11. Given an interaction $A \rightarrow B$, the roles of A and B are R_A and R_B respectively. We assume that roles have integer values and those closer to the root have greater values than those closer to the leaves. The distance between A and B is $|R_A - R_B|$. For example the distance between the president and a professor is 2, while the distance between the deans of the math and the arts faculties is 0. If A is higher than B in the role tree, the importance should be increased, otherwise

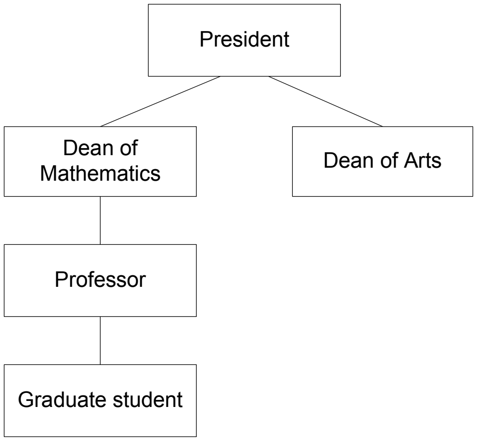


Figure 2.11: A sample role tree for a university

decreased. The new importance should still be in $(0,1]$.

In principle, the importance modifier based on the role relationship is a function $f(I_A, R_A, R_B) \in (0,1]$, where $I_A \in (0,1]$. Clearly, some exponential functions of $R_A$and R_B have this property. Linear functions can also be applied with extra treatment of boundary conditions. The following is an example of a linear modifier.

$$f(I_A, R_A, R_B) = \begin{cases} \min((R_A - R_B) \cdot I_A, 1) & if\, R_A > R_B \\ \frac{I_A}{R_B - R_A} & if\, R_A < R_B \\ I_A & if\, R_A = R_B \end{cases}$$

One can also imagine other classes of modifiers. Among all possible choices, we prefer exponential modifiers because of their intrinsic commutative property. Algorithm 2.2 is a modifier that we propose as an example of a class of exponential modifiers.

Consider the student-instructor example again. The role distance between a student and an instructor is 1 according to the role tree in Figure 2.11. Suppose the importance of the interaction is 0.8. It is high, because the student is a little bit desperate. Suppose the instructor's willingness is 0.3, which means he is busy working, so please do not disturb if not urgent. In this case, the importance threshold is $1 - 0.3 = 0.7$. Without the modifier, the professor would face an intrusion due to the exaggerated importance set by the student. If we apply the algorithm, the new importance becomes $0.8^{1+1} = 0.64$, which is lower than the instructor's importance threshold, which is also the unwillingness, and the interaction is prevented. One might argue that if the student sets the original importance to 0.9, the new importance would be 0.81, which would let the intrusive interaction hap-

Algorithm 2.2 Importance modifier for roles

```
/*
 * given the original importance and a role tree,
 * this algorithm returns the new modified importance
 *
 * $R_A$, $R_B$: the integer level of $A$ and $B$ in the tree
 * $R_A > R_B$ if $A$ is at a higher level than $B$
 */
if the sender or the receiver is not on the role tree
  return $I_A$;
else if $R_A > R_B$,        /* increase $I_A$ */
  $newI_A \leftarrow I_A^{\frac{1}{(R_A - R_B)+1}}$;
else                         /* decrease $I_A$ */
  $newI_A \leftarrow I_A^{(R_B - R_A)+1}$
return $newI_A$;
```

pen. In a closed environment, there are limited kinds of interactions, and users usually have some shared understanding of the importance of interactions. We also assume that users will not misrepresent importance, as they know each other. Besides, at the system level (see Chapter 5), we use the IM presence model to implement the comparison filtering, which asks the receiver to approve the sender's presence-subscription request. It is easy for the sender to deny requests from those who exaggerate importance. Furthermore, the sender can block communication completely from selected users if they are aggressive. In this way, we can deal with malicious users who exaggerate importance and try to interrupt others with "spam" interactions.

This example shows that an appropriate modifier for roles can reflect importance more precisely. Users may choose different classes of functions for modifiers. However, we are focusing on demonstrating the model and ideas, rather than on finding or proposing the best function. Next, we describe an example atomic modifier for willingness.

In our model, the receiver sets a general willingness for all interactions and all senders, although this general willingness changes with time and location at the receiver's discretion. An alternative is to use an individual willingness for each interaction and each sender. We consider the latter infeasible in a pervasive- and ubiquitous-computing environment, as there are many entities and interactions, and the receiver cannot foresee what interactions will arrive. A general willingness, however, might block some interactions that are actually wanted by the receiver. For example, Bob is in a meeting discussing new equipment to be purchased. Bob presents a low willingness, as he does not want other people to disturb him. During the meeting, Bob's secretary Alice tries to interact with him about some price information she just found for the equipment. If the willingness is not modified to suit this situation, this interaction may not happen, because it is not an emergency, so it will not have a high importance. The role relationship between Alice and Bob may further reduce the probability of this interaction happening, as it decreases the importance of the interaction. However, intuitively, Bob does want this interaction, because it is related, and is helpful for his current task.

So in general, we assume an entity presents a willingness that is suitable for most of the interactions. However, there are usually some exceptions, such as when Bob wants to interact with Alice if the interaction is related to the meeting. A receiver should provide such information as the subject of his current activity. A willingness modifier takes the original willingness, and generates a new one based on the extra information that the receiver provides. We call this a willingness modifier for exceptions.

We describe one importance modifier for roles, and one willingness modifier for exceptions. We choose them as they are obvious and match intuition. However, there could be many such atomic modifiers for either the importance or the willingness. For example, we can suppose that in a company, interactions on Monday morning have a higher importance than the rest of the week, as important decisions or meetings usually happen on Monday morning in this company. So in this case, we need an importance modifier for time. Similarly, we might need a willingness modifier for location, weather, and so on.

Each of these modifiers does the same thing, which is refining the importance or the willingness so they can fit the situation better and the model can generate more accurate results. At the same time, each modifier is based on some context of the system. For example, an importance modifier for time obviously requires the time information. A willingness modifier for location requires the location information to be available. Obtaining some context might be easy, such as the time, or might be more difficult, such as the location. The availability of the context information determines the usability of modifiers. Once we have new context, we can add modifiers accordingly.

Suppose we have more than one modifier for either importance or willingness. For example, we can have two importance modifiers, one for role relationships and the other for time. They are going to take effect independently. That is to say they can both increase or decrease the importance, or one could increase the importance while the other decreases it. If both modifiers are exponential, then they are commutative. Thus the order of application does not matter. In general, we prefer commutative modifiers. Other refinements can also be considered. For example, a user might think that modifier X should be three times more effective than modifier Y. We exclude this, as well as how to make commutative modifiers, from our model at this time. Instead, we leave them to future work.

The composition of atomic modifiers has a prerequisite, which is that the context they use should be orthogonal so that modifiers are independent of each other. Otherwise, they cannot simply be connected into a pipeline without considering their correlations, which is the complexity we want to avoid. However, this is not always the case, since, for example, the location and the activity may be related. Or, it just might be more suitable to consider a variety of context together to infer the new importance or willingness. Thus it is necessary to have complex modifiers in addition to atomic ones.

2.3.7 Complex modifiers

A complex modifier takes multiple context as input and produces an inferred value of I_A or W_B. Figure 2.12 is an example, which infers willingness using four pieces of context, i.e., the IM status, the calendar, the status of the office door, and the time of day. The internal logic of such modifiers can be based on rules, such as "if the IM status is online, the current calendar is empty, the door is open, and the user is on lunch break, then the user's willingness is 0.9," or can use a more comprehensive mechanism, such as a Bayesian network [57]. However, finding the optimum mechanism for a complex modifier is not the focus this thesis. Overall, complex modifiers are tightly bound to users and applications and are less general than the atomic ones, but appropriate when different context is related or the pipeline approach does not fit well.

We may combine atomic and complex modifiers to form a hybrid approach, so that we can utilize the advantages of both and incorporate a variety of context. Next we refine the comparison model to include context and modifiers.

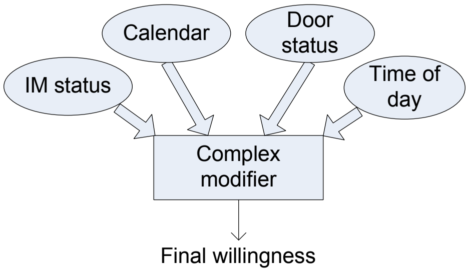


Figure 2.12: Complex modifier for willingness

2.3.8 The comparison model with modifiers

The original comparison model in Section 2.3.3 compares raw importance and willingness to determine if an interaction is intrusive. In the refined model shown in Figure 2.13, both I_A and W_B go through a series of modifiers and receive new values for them, and then the new importance is compared to the new importance threshold to decide the intrusiveness of an interaction. Note that all modifiers require the input of some context. The complexity of these modifiers depends on the availability of context.

We have described how a sender can determine the deliverability of an interaction during the filter stage. The importance of the interaction is compared against the importance threshold or the unwillingness of the receiver. We use context to modify the importance and the importance threshold so they are more accurate. Next we move to the delivery stage.

2.4 Non-intrusive computing: Delivery stage

In this stage, we know that the sender has decided the interaction will not intrude upon the receiver. So it is taking place and should be delivered in such a way that it catches the receiver's attention, which we model with *effectiveness*. At the same time, an inappropriate delivery may interrupt or intrude upon other users in the same environment, such as when delivering a personal conversation to a phone that rings while people are having a meeting. We use *overtness* to model this factor. Obviously, we want delivery to have low overtness, yet high effectiveness.

A pair consisting of a device and one of its notification methods is called a delivery candidate. The delivery stage starts from a collection of known candidates, as shown

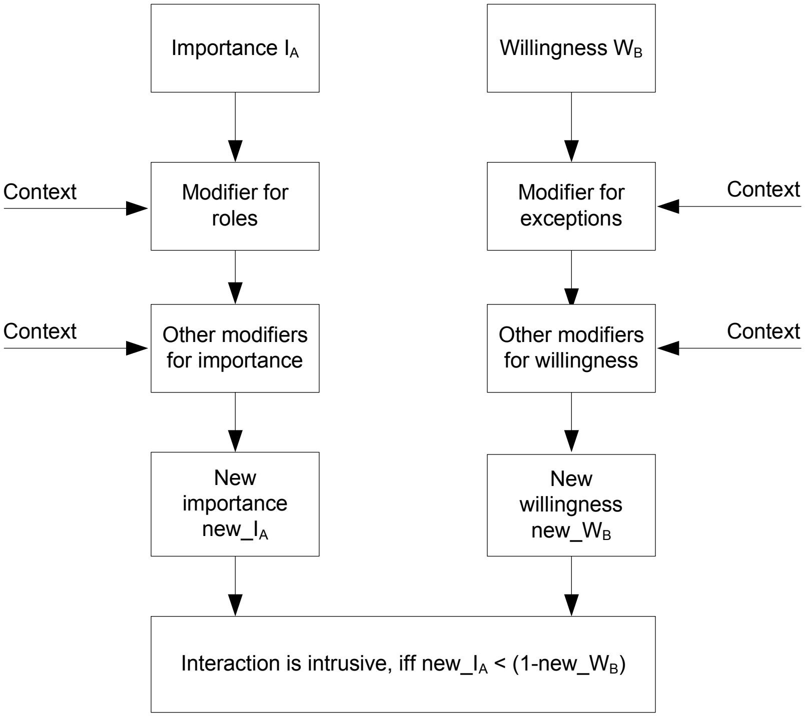

Importance I_A
Willingness W_B
Context
Modifier for roles
Modifier for exceptions
Context
Context
Other modifiers for importance
Other modifiers for willingness
Context
New importance new_I_A
New willingness new_W_B
Interaction is intrusive, iff new_I_A < (1-new_W_B)

Figure 2.13: The comparison model with modifiers

in Figure 2.14. Then we use context to find feasible ones. For example, we may use location information to find only those devices that are in the same location as the user. A detailed discussion of this step is in Section 2.4.1. With feasible candidates, we apply the effectiveness and the overtness criteria (Sections 2.4.2 and 2.4.3) to ensure effective delivery with tolerable overtness. In the end, there are one or more appropriate candidates that will be chosen for delivering the interaction to the receiver.

2.4.1 Feasibility criteria

We know that an interaction should be delivered to the receiver in an effective yet unobtrusive manner. Before we discuss these in detail, we describe a few other criteria that eliminate infeasible delivery candidates. For example, any selected candidates have to be at the same location as the receiver, otherwise, the receiver will miss interactions delivered to them. For another example, delivery candidates should be available when the interaction happens. Suppose we decide to redirect a message to a landline phone, but it is being used by another user. Obviously, we should consider this case and avoid it. Figure 2.14 shows four such criteria, each of which eliminates infeasible candidates

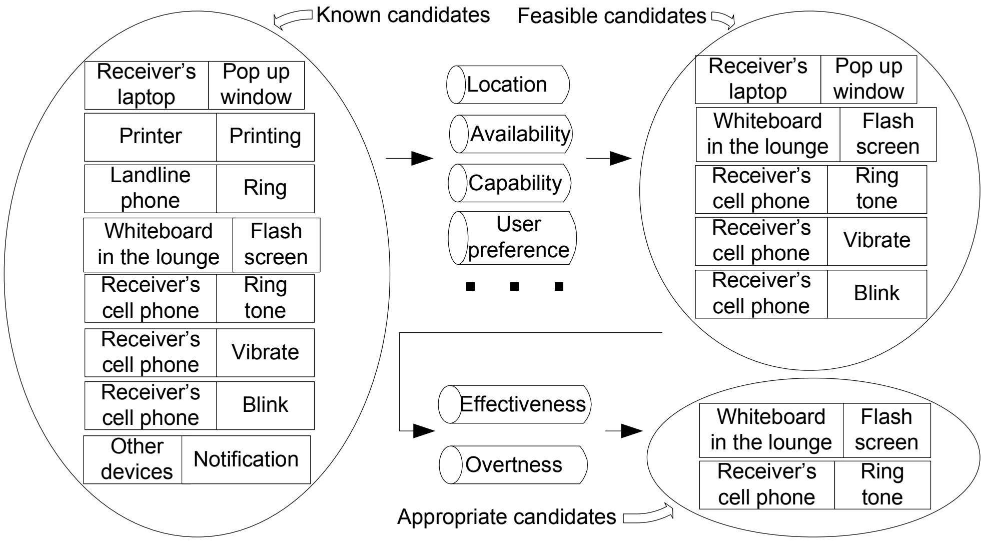


Figure 2.14: Conceptual view of the delivery stage

in some way.

- Location: Candidates should be at the same location as the receiver.
- Availability: Candidates should be available when the interaction happens.
- Capability: Candidates should be capable of handling the interaction, such as a display and a speaker for multimedia interactions.
- Preference: Users may have preferences in selecting candidates, and these should be respected if possible.

All these criteria depend on some context whose complexity varies. For example, the context required by the capability criterion is easy to obtain, assuming it depends only on static characteristics of the device. For another example, imagining there exists a criterion based on the physical modeling of a space, it would be difficult for a system to provide this type of context, because the physical modeling requires deployment of large number of sensors or cameras, which many practical systems do not yet have. Nonetheless, based on how much context is available, there may be more or fewer such criteria than in Figure 2.14 . The more context we have, the more we are able to make good choices of delivery candidates.

After applying feasibility criteria, the remaining candidates are subjected to the effectiveness and overtness criteria, as shown in Figure 2.14. The next two sections discuss them in detail.

	Audible		Visual		Motion		Nothing	
	E	O	E	O	E	O	E	O
Workstation	0.5	0.5	0.3	0.3	n/a	n/a	0	0
Cell phone	0.7	0.7	0.4	0.4	0.6	0	0	0
Printer	0.6	0.6	n/a	n/a	n/a	n/a	n/a	n/a
whiteboard	n/a	n/a	0.8	0.8	n/a	n/a	0	0

Table 2.2: An example of assigning effectiveness (E) and overtness (O) for a few devices and notification methods

2.4.2 Effectiveness

A pervasive-computing environment is usually assumed to be saturated with devices for users to interact with each other or with the rest of the environment. Each of these devices may have one or more choices in terms of capturing a user's attention, as we mention briefly in Section 2.2. Different notification methods of the same device or different devices may imply different levels of effectiveness when it comes to notifying users. For example, a cell-phone ringing may be more effective than blinking. As another example, a conventional land-line phone ringing may be more effective than a cell-phone ringing, as the former usually rings louder.

Although each individual notification method is different, they can be divided into four categories, which are audible, visual, motion, and doing nothing. Table 2.2 is an example of assigning effectiveness values for each of these categories for a few devices. We can see that the values for effectiveness range between 0 and 1, inclusive. These values can be interpreted in two ways. On the one hand, zero means that the notification does nothing actively, and the user will have to take independent action to perceive the interaction. Value 1 means that the notification is effective enough that the user should notice immediately. On the other hand, these values can be thought of as the probabilities of the receiver perceiving the notification. The higher the effectiveness value for the notification, the more likely the user noticing the interaction.

Let D be a device, and N be one of D's notification methods, then $c = (D,N)$ is a delivery candidate. $E(c) \in [0,1]$ is the effectiveness of c. We also have $E(c)$ is the probability of user perceiving the notification when delivering an interaction using c. We use similar notation for overtness in Section 2.4.3.

It is possible to combine multiple delivery candidates to achieve a higher effectiveness. For example, we consider that a cell-phone ringing at the same time as a land-line phone is more effective than each of them ringing alone. However, it is not always possible to combine multiple delivery candidates. For example, there are three delivery candidates, the land-line phone ringing with high volume, the land-line phone ringing with medium volume, and the land-line phone ringing with low volume. They have the same device but different notification methods. In this case, we are not able to combine any two of these three candidates to achieve higher effectiveness. To solve this problem, for a device with multiple notification methods that may be exclusive to each other, we add all non-exclusive combinations of notification methods as delivery candidates. When the system tries to select multiple delivery candidates, it cannot select more than one with the same device.

Now suppose two delivery candidates are compatible. The combined effectiveness is defined as the probability of the user perceiving the notification when using either or both candidates. Let c_1 and c_2 be two delivery candidates, $E(c_1 \cup c_2)$ the combined effectiveness of them, and $E(c_1 \cap c_2)$ the joint effectiveness of the two. Based on probability theory, we have

$$E(c_1 \cup c_2) = E(c_1) + E(c_2) - E(c_1 \cap c_2)$$

We assume delivery candidates are independent in terms of capturing users' attention. Thus $E(c_1 \cap c_2) = E(c_1)E(c_2)$. Hence,

$$E(c_1 \cup c_2) = E(c_1) + E(c_2) - E(c_1)E(c_2)$$

Generalizing, we can model the union effectiveness of three or more independent delivery candidates.

$$\begin{aligned} E(c_1 \cup c_2 \cup c_3) &= E(c_1) + E(c_2) + E(c_3) - E(c_1)E(c_2) - E(c_2)E(c_3) \\ &\quad - E(c_1)E(c_3) + E(c_1)E(c_2)E(c_3) \end{aligned}$$

In this way, we are able to combine as many delivery candidates as possible in order to be "effective enough." We expand the above effectiveness notation for a single candidate to a set of candidates.

Let $\sigma = \{c_1, c_2, ..., c_n\}$,assuming $c_1, c_2, ..., c_n$ are compatible, we have $E(\sigma) = E(c_1 \cup c_2 \cup ... \cup c_n) \in [0,1]$.

Note that we currently make an assumption of independence, but also refine the implementation to mark candidates as mutually exclusive if they cannot be combined. Also, it is possible to completely eliminate this assumption by using fuzzy logic. It defines effectiveness of each candidate in fuzzy sets, and then uses rules to infer the combined effectiveness. This approach requires us to define rules for any possible combination of available candidates, which will result in a large number of rules. This is not the focus of this paper, and we will investigate this issue in the future.

Intuitively, if an interaction is important enough, or the receiver's willingness is very high, we should make sure that the receiver does not miss this interaction. This means we should use a very effective notification. Similarly, if the importance is not very high but higher than the importance threshold, we should deliver the interaction effectively but without overwhelming the receiver. This suggests that the effectiveness of the chosen delivery candidates should be related to the deliverability, as described in the filter stage.

Let S be the set from which we choose delivery candidates. SE is a set of sets, each member in SE is a subset of S, whose effectiveness should be greater than or equal to the deliverability calculated in the filter stage. and satisfies the condition below, i.e.,

$$SE = \{\sigma \subset S \,|\, E(\sigma) \geq d = I_A - \overline{W}\}$$

We call this the *effectiveness criterion*, which is that an interaction should be delivered to those devices with certain notification methods so that the combined effectiveness is greater than or equal to the difference between the importance I_A and the importance threshold, denoted by $\overline{W}$ ($\overline{W} = 1 - W_B$). In the context of effectiveness, the difference of I_A and $\overline{W}$ represents the net importance of the interaction, and we want the delivery effectiveness to outweigh it.

This effectiveness criterion only benefits the receiver. However, as the receiver often shares an environment with other users, we should consider them as well. In the next section, we discuss how to make sure our delivery choice does not cause intrusion problems for other users in the same physical environment.

2.4.3 Overtness

The previous discussion is based on the observation that devices and notification methods imply different levels of effectiveness. Similarly, each delivery candidate has some degree of overtness. For example, a cell-phone ringing has a higher overtness than vibration. For another example, popping up a window on a big whiteboard has an even higher overtness than a cell-phone ringing in a class. Again, Table 2.2 is an example that lists possible overtness values for a few devices and notification methods. Overtness values also range from 0 to 1.

Let $c = (D,N)$ be a delivery candidate, we have $O(c) \in [0,1]$, which is also interpreted as the probability that some other user will be aware of the delivery through c. We now expand this notation to a set of candidates, as in Section 2.4.2. Suppose $\sigma = \{c_1, c_2, ..., c_n\}$, then $O(\sigma) = O(c_1 \cup c_2 \cup ... \cup c_n) \in [0,1]$.

We want to point out that some delivery candidates have the same value for effectiveness and overtness, such as a workstation with audible notification, while others have zero overtness, such as a cell phone with motion, i.e., vibrating. This is because some delivery candidates have the same notifying effect to all people. For those with non-zero effectiveness but zero overtness, they tend to notify the receiver only, thus causing no overtness to others. Section 2.4.4 has more details on the relationship between effectiveness and overtness.

Depending what delivery candidates we choose, the overtness should be low and acceptable. Each environment has an indication of its overtness requirement. For example, a meeting situation has a strict requirement on overtness, i.e., delivery overtness should be very low. For another example, a lunch-break situation has a loose overtness requirement where everybody can make more noise or move around. We use OT (overtness threshold) to denote the requirement imposed by the environment. The range of OT is $[0,1]$. A lower OT means a more strict overtness requirement. When OT becomes 0, it means right now there should be no overtness at all. Table 2.3 assigns OT values for a few example scenarios, based on the user study in Section 2.5. The values are linearly mapped from the majority answer to each question. For example, in the case of shared lab, the majority of the participants chose “neither high nor low.” As another example, in the case of being alone, the majority chose “very high.” More details on this regard is in Section 2.5.

Using notation similar to Section 2.4.2, we have the following *overtness criterion*.

	Overtness Threshold (OT)
meeting in a meeting room	0.25
working in a shared lab	0.5
lunch break in a lounge	0.75
being alone in a private room	1

Table 2.3: Overtness thresholds based on the user study in Section 2.5

$$SO = \{\sigma \subset S \,|\, O(\sigma) \leq OT\}$$

It says that the combined overtness of a set of delivery candidates should be less than the overtness threshold, which is predetermined based on the environment context.

We have described the effectiveness and overtness criteria, which are fundamental in the delivery stage. Although effectiveness and overtness measure a delivery candidate's impact on the receiver and other users respectively, they are related in some circumstances. Section 2.4.4 explores the relationship between them, to facilitate the selection of delivery candidates in Section 2.4.5.

2.4.4 Relationships between effectiveness and overtness

Both effectiveness and overtness characterize how a delivery method may affect people. Effectiveness evaluates a delivery candidate in favor of the intended receiver, while overtness is a side effect to unintended users. Their relationship is two-fold. On the one hand, if a delivery candidate has a high overtness, it tends to also have a high effectiveness because the receiver is among all users who perceive the interaction. For example, if a private message is delivered to a whiteboard while the receiver and other users are using the whiteboard in a meeting, all of them will notice the message. On the other hand, a low overtness does not necessarily indicate low effectiveness, as the effectiveness can be confined to the receiver. For example, a cell phone vibrating can easily capture the receiver's attention, but not that of others.

Delivery candidates can be divided into two groups, one of which has effects on all users when delivering an interaction to a receiver, such as a whiteboard displaying or a phone ringing, the other of which has effects only on the receiver, such as a cell-phone vibrating or a voice notification from a pair of earphones, assuming the receiver has the

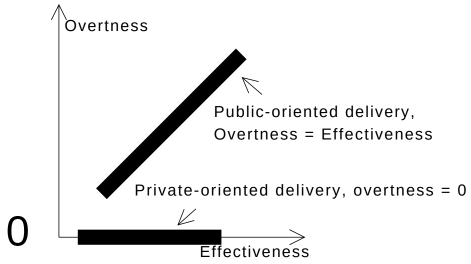

Overtness
Public-oriented delivery,
Overtness = Effectiveness
Private-oriented delivery, overtness = 0
0
Effectiveness

Figure 2.15: Relationship between effectiveness and overtness

cell phone in his pocket, or is wearing earphones. We call these two groups *public* and *private delivery* respectively. Each delivery candidate (a pair of device and notification method) belongs to one of these two groups.

Since public delivery candidates affect all users, the effectiveness value to the receiver is close to the overtness value to other users. As private delivery candidates only affect the intended receivers, their overtness is low and negligible. We assume that a public-delivery candidate has the same value for effectiveness and overtness; a private-delivery one has zero overtness value. In reality, effectiveness and overtness may not be exactly the same for public delivery and overtness may not be completely negligible for private delivery. For example, a message on a whiteboard may not be easily read by people who do not have good angles. For another example, although cell-phone vibration is considered private delivery, it may capture all users' attention, because this cell phone may be on a table instead of in somebody's pocket, or the vibration may be significant enough that people nearby can still hear it even though the cell phone is in the receiver's pocket, or some people further away may have good hearing. However, we consider these discussions beyond our focus and leave them to future work.

Figure 2.15 reflects public and private delivery with regard to overtness and effectiveness. The bar on the x axis represents private delivery candidates with zero overtness; while the inclined bar represents public delivery candidates whose overtness and effectiveness are always the same.

As we mention briefly at the the end of Section 2.4.3, the relationship in Figure 2.15 helps reduce the complexity in choosing delivery candidates. Section 2.4.5 discusses the selection process in detail.

2.4.5 Delivery-candidate selection

The aim of the delivery stage is to select proper candidates. We discuss selection criteria from Sections 2.4.1 to 2.4.4. As shown in Figure 2.14, the feasibility criteria are applied first, then are the effectiveness and overtness ones. Applying the feasibility criteria is straightforward given the necessary context. Depending how this context is provided and accessed, we may apply the feasibility criteria as a single query to the context system, if complex querying is allowed, or a sequence of queries, otherwise. In the case of sequential queries, the order does not matter, as each one eliminates infeasible candidates. The result is the feasible set, FS. For now, we assume the existence of the underlying system and its context management facilities in order to carry on the selection process. The detailed description of the system is in Chapter 5.

When the effectiveness and overtness criteria are applied to FS, each produces a set of sets, *SE* and *SO*; hence their intersection, *RE*, is also a set of sets. Conceptually, every member in *RE* is a potential final choice containing one or more delivery candidates.

$$\begin{aligned} SE &= \{\sigma \subset FS \,|\, E(\sigma) \geq d = I_A - \overline{W}\} \\ SO &= \{\sigma \subset FS \,|\, O(\sigma) \leq OT\} \\ RE &= SE \cap SO \end{aligned}$$

Applying SE and SO to FS requires combined overtness of RE less than OT, and the combined effectiveness of RE greater than the net importance d. This problem is essentially a knapsack problem [62], where there is a set of items, each of which has a value and a cost. The problem is to determine a collection of items whose total cost is less than a given limit, i.e., the capacity of the knapsack, and the total value is as large as possible. In our case, the overtness threshold is the cost limit. The effectiveness of a candidate is the value of an item, and the overtness is its cost.

The difference is that in the knapsack problem, the cost or the value of multiple items is the sum of individuals, whereas in the candidate-selection problem, the effectiveness or overtness of multiple candidates is the combined probability of individuals. Also, the selection problem has a lower bound constraint on the effectiveness, the net importance, while the knapsack one does not. Despite these differences, the selection problem is the $0-1$ knapsack problem, which is NP-complete [62]. Dynamic programming [59] can be used to solve these problems [62]. However, it requires much space to compensate for the run-time complexity, which may be a constraint on mobile devices, where the selection is computed.

In Section 2.4.4, delivery candidates are divided into two groups, public and private. Based on the discussion of the relationship between the effectiveness and the overtness, we assume that for public delivery candidates, the effectiveness is the same as the overtness, and for private ones, the overtness is zero. These relationships will not change the nature of the problem in the worst case, however, they can simplify it in many situations.

The strategy is that the private delivery candidates are always selected. If their combined effectiveness is greater than the lower bound of the effectiveness required in SE, i.e., $I_A - \overline{W}$, the problem is solved, as these candidates have zero overtness. Otherwise, we compute the remaining effectiveness requirement by removing that combined effectiveness. It then becomes the new lower bound for the effectiveness criterion.

The problem now is to find a set of public-delivery candidates with union overtness less than OT and union effectiveness greater than the new lower bound. In general, this is a knapsack problem. However, there is is a special case where there may be at least one public-delivery candidate whose effectiveness or overtness is in between the lower bound and the upper bound. In this case, we only need to check each public-delivery candidate in linear time. Besides, we may use exhaustive search to deal with NP-completeness. In reality, it is rare to have more than, say, 20 feasible delivery candidates at one time in a smart space, so exhaustive search is viable. This process is also described in the Algorithm 2.3.

Note that when the effectiveness and overtness criteria, i.e., SE and SO, cannot both be satisfied, we can relax the feasibility criteria so that we enlarge the input set of delivery candidates, i.e., FS, and hope appropriate candidates are found. It is possible that even if FS is relaxed and larger, SE and SO still cannot be satisfied at the same time. Or we may not want to relax FS at all. In those cases, SE or SO has to be overridden. The system preference may specify rules in this regard. Also, Algorithm 2.3 tends to return multiple delivery candidates, which may cause complexity at the system level, as the interaction will have to be delivered to multiple candidates. Hence, we may add some constraints to the problem, such as the result size should be no larger than two. While this reduces the complexity of the problem, it may decrease the probability of finding appropriate candidates. However, this level of detail is of concern for a specific system implementation and is not the focus of this dissertation.

Algorithm 2.3 Delivery-candidate selection, part 1

```
/*
 * this algorithm selects delivery candidates
 * from the feasible set
 * if it returns null, then
 *   option 1: relax the feasibility criteria
 *   option 2: override criteria $SE$ or $SO$
 * if it becomes the knapsack problem, then
 * use dynamic programming or exhaustive search
 * if the number of candidates is small
 *
 * input: $FS$, $PrDS$, $PuDS$, $d$, and $OT$
 *   $FS$: feasible set of candidates
 *   $PrDS$: set of private-delivery candidates
 *   $PuDS$: set of public-delivery candidates
 *   $d$: net importance($d = I_A - \overline{W}$)
 *   $OT$: overtness threshold
 *
 * output: $RE$
 *   $RE$: set of appropriate candidates for delivery
 */
$RE \leftarrow PrDS$        /* all candidates in PrDS are returned */
$lowBound \leftarrow \frac{d - E(PrDS)}{1 - E(PrDS)}$ /* new lower bound for effectiveness */
if $lowBound < 0$  /* PrDS is effective enough */
  return $RE$
if $lowBound > OT$  /* cannot satisfies both $SE$ & $SO$*/
  relax feasibility criteria, or
  override $SE$ or $SO$

/* continued on next page */
```

Algorithm 2.4 Delivery-candidate selection, part 2

```
/* continued from previous page */

sort $PuDS$, high overtness first
backup $PuDS$
if $\min(PuDS) > OT$   /* $SO$ cannot be satisfied */
  relax feasibility criteria, or
  override $SE$ or $SO$
if $\max(PuDS) < lowBound$
  dynamic programming or exhaustive search
$c \leftarrow$first element in $PuDS$
while $|PuDS| > 0$ and $!(lowBound < O(c) < OT)$
  $PuDS \leftarrow PuDS - \{c\}$
  $c \leftarrow$first element in $PuDS$
if $|PuDS| > 0$
  return $RE \cup \{c\}$
else
  dynamic programming or exhaustive search
```

2.5 User study

In order to investigate how users understand the model and their reaction to it, we conducted a user study using a survey, where users are presented with hypothetical interactions and answer questions about the concepts of importance, willingness, effectiveness, overtness, and overtness threshold. A demographic question is asked in the beginning of the survey regarding the familiarity with various computing devices.

The survey has four parts, each dealing with one or two parameters in the model. Each part has five to eight questions. For example, in the willingness part, we ask users their general willingness to interact with other people when they are discussing a project with colleagues in a meeting. As another example, in the overtness threshold part, we ask users to set the threshold values in different scenarios, such as a meeting, a job interview, and so on. Answers to all questions use a 5-level Likert scale [63]. We also give them options to choose not to answer the question, as shown in Table 2.4.

We recruited 22 participants in two weeks, all being UW students or researchers from

○very high	○ high	○ neither high nor low	○ low	○very low
○no opinion				

Table 2.4: Answers to the questions in the survey

different faculties and departments. In each interview, we gave a brief introduction to this study and then the participant finished the survey independently, either on site or off site. It takes, on average, about 20 minutes to answer all questions.

For the demographic question, all participants considered themselves familiar or very familiar with computer devices. One participant selected "no opinion" in one of the questions regarding the importance; one participant chose "no opinion" in one of the questions regarding the effectiveness and overtness of delivery candidates.

For the statistical results, we choose values, 5, 4, 3, 2, and 1 for answers from "very high" to "very low" respectively. Table 2.5 shows the averages and the standard deviations for different scenarios where participants are asked about their willingness. Table 2.6 is for importance. Table 2.7 lists a number of delivery candidates and their effectiveness and overtness. Table 2.8 shows the results for the overtness threshold.

The willingness averages of different questions are considered normal. For example, the willingness average of the meeting scenario (Table 2.5) is 2.14, which is between "neither high nor low" and "low," and closer to "low." The average for the coffee break one is 4.14, which is consistent with intuition. Some standard deviations are relatively high. In the meeting case, the deviation value is 0.92. Part of the the reason for this is that participants were considering more information than is provided on the survey, for example, whether interactions are related to the meeting. However, the majority of participants still show consistency to some degree. In Figure 2.16, three people chose 4 ("high") when asked their willingness in a meeting; two people chose "neither high nor low"; 17 out of 22 (77%) participants chose "low" or "very low."

We find that values for importance tend to be more stable, i.e., smaller standard deviations compared to those for willingness. At the same time, the average values are larger than those for willingness. There are five scenarios regarding the importance (Table 2.6), four of which have values greater than 4. Although these scenarios are not unimportant, there may be some endowment bias [60], which, in this case, means that people consider their own interactions more important than others. It is difficult to completely avoid this, if we let individuals determine importance values. In Section 2.3.6, we introduce modi-

Scenario	Average	Standard deviation
You are discussing a project with colleagues in a meeting. What is your willingness to interact in general?	2.14	0.92
You are having coffee during a break in the lounge where people gather and chat. What is your willingness to interact in general?	4.14	0.69
You are a faculty member working on your research in your office. A student shows up and wants to talk about the final exam with you. It is not your regular office hour. What is your willingness to interact with the student?	2.82	0.98
You are home on the weekend, relaxing. What is your willingness to interact in general?	3.82	1.23
Suppose you have many important business emails, correspondence, orders, etc, to deal with on Monday morning, presumably accumulated over the past weekend. You are processing them. What is your willingness to interact in general?	1.95	1.11

Table 2.5: Statistical results for willingness

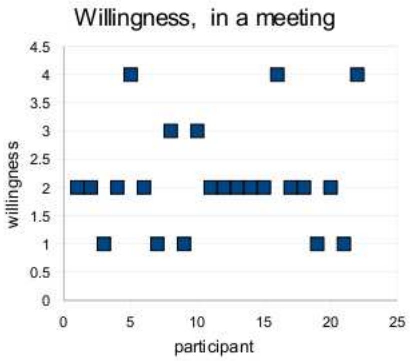


Figure 2.16: Willingness to interact when in a meeting

fiers and discuss how they may refine the initial values. An example is that a student may exaggerate the importance of an interaction with an instructor, in which case the system will then reduce the value based on the role relationship. Hence, the endowment effect can be rectified by different modifiers.

As to the delivery candidates, according to Table 2.7, the landline phone and cellphone ringing are the most effective and the most overt. The cellphone blinking and the printer printing are the least effective and overt.

In Section 2.4.4, public and private deliveries are discussed. The approximation is that for the public delivery, the effectiveness is the same as the overtness; for private delivery, the overtness is negligible. According to the survey, the effectiveness is slightly higher than the overtness. One reason is that users are used to the disturbance and are generally immune to noise from the environment. Another reason is that, in the case of the cellphone ringing, the owner is more sensitive to the ring tone. Hence, in terms of capturing people's attention, a particular ring tone is more effective to the person who selects it than to others.

For the private delivery, the overtness is not zero in Table 2.7, which is expected. Given more context, such as how close one user is to another when the cellphone vibrates, the system is able to assign more precise values for overtness.

We also calculate the correlation coefficient between the effectiveness and the overtness. The result (Table 2.7) shows that these two are barely correlated. This confirms the fact that effectiveness and overtness are indeed two different metrics, as one is for the intended receiver, and the other is for the unintended people nearby. One does not necessarily determine the other.

For the overtness threshold (Table 2.8), the chatting scenario has a high threshold (average: 4.23), as does the user being alone, with an average of 4.41. The meeting scenarios have a lower threshold value. It is unanimously agreed that in a private job interview, there should be no overtness (average: 1, standard deviation: 0).

From this user study, we understand how users react to various scenarios. On the one hand, it shows that the majority of the participants tend to make similar or the same choices, which helps a system infer and determine the parameters in the model. On the other hand, it implies that we need much context in order to make appropriate decisions. People could have very different opinions occasionally, which suggests that before a system is deployed, it needs to take users' preferences into consideration. Also, it confirms

Scenario	Average	Standard deviation
You are involved in a project. You need frequent discussions with other people about your solutions, progress, problems, and so on. Now you try to initiate such a discussion. What is the importance of this discussion?	4.18	0.49
You witnessed an elderly neighbor fall in front of his house, you helped the neighbor and called an ambulance and now you want to call his son at work. What is the importance of this call?	4.73	0.45
During work time for both of you, you want to invite a friend to a party using an instant message. What is the importance of this message?	2.55	0.94
You are a project coordinator. You need to organize a meeting and want to call a participant for his schedule. What is the importance of this call?	4.05	0.64
Suppose you are the coordinator in the previous scenario. One participant has not confirmed his attendance at the meeting by the deadline. You want to go over to his office and knock on his door. What is the importance of this?	4.48	0.5

Table 2.6: Statistical results for importance

Delivery candidate	Effectiveness		Overtness		
	average	standard deviation	average	standard deviation	coefficient
landline phone, volume high	4.68	0.47	4.36	0.83	0.3
landline phone, volume low	3.43	1.05	2.95	1.09	0.6
cellphone, ringing	4.55	0.58	4.09	0.9	0.17
cellphone, vibrating	3.95	0.88	1.5	0.72	-0.04
cellphone, blinking	2.41	1.07	1.27	0.62	0.45
shared printer, printing	2.41	0.98	2.23	0.9	0.05
whiteboard, displaying	3.41	0.89	2.82	1.3	0.3
laptop, popping up window	4.27	0.81	1.55	0.94	0.04

Table 2.7: Statistical results for effectiveness and overtness

Scenarios for the overtness threshold	Average	Standard deviation
Small meeting involving at most three people	1.91	0.67
Large meeting involving more than ten people	1.82	0.83
You are the only person in the office	4.41	0.72
A typical computer lab, shared with a number of other users	2.95	0.77
People chatting in a lounge during lunch time	4.23	0.6
In a technical presentation	1.14	0.34
As the prospective employee in a private job interview	1	0

Table 2.8: Statistical results for overtness threshold

that the assumptions we make regarding the public and private delivery are close approximations. In reality, the effectiveness of a public delivery candidate is higher than the overtness; and the overtness of a private delivery is close to, but not, zero. In a practical system, we can set default values for the parameters in the model, such as what we did in Table 2.3. Users should be able to override them at any time. Furthermore, the system may needs to be customized or trained for individual users.

2.6 Summary

We have analyzed and modeled the intrusion problem associated with a time- and reference-coupled interaction, including deciding whether it is deliverable, and if it is, how to choose devices and how to notify the receiver. The model is general and applicable to different interactions. For example, we may use it to enhance the low-battery warning system on a laptop computer. Currently, for laptop users, if the remaining battery power is less than a predefined amount, the system notifies users by popping up a dialog window or showing a message, which may cause some interruption to users. Besides, it always uses the same means to alert users, which could be inappropriate in some situations. Our model can be applied to address these issues.

The model is divided into the filter and delivery stages. In each one, key elements are identified, i.e., importance and willingness in the filter stage, and effectiveness and overtness in the delivery stage. The elements are quantified to enable computation and comparison. These two stages are connected internally by requiring the effectiveness of selected delivery candidates in the delivery stage to be greater than or equal to the net importance in the filter stage.

In the filter stage, we compare the importance of an interaction with the willingness of the receiver, to determine the deliverability. Continuing the battery-warning example, the remaining power indicates the importance of a notification. The lower the power level, the higher the importance. The result will be that unless the battery is critically low, no notification will be delivered to a busy user with a low willingness.

In the delivery stage, several criteria are proposed. Feasibility criteria can identify devices and notifications that are not suitable for delivery for various reasons, such as not being co-located with the receiver, not being capable of handling the interaction, and so on. Then the effectiveness and overtness criteria are applied to ensure that the receiver notices the interaction and delivering the interaction will not disturb other users.

In the battery-warning example, there could be many different ways to inform users, depending on available context. For example, if a user is presenting a research paper, the warning software may send a message to his cell phone which vibrates. This avoids popping up a message on the projector screen. As another example, if a user is reading a hard-copy document, it is better to play a soft tone as he is not looking at the screen. Furthermore, if a user is temporarily away from the laptop, the notification could be redirected to a close land-line phone or to his cell phone, assuming an indoor location-tracking system and the availabilities and settings of those devices involved. In this way, this low-battery-warning system will be less intrusive to users, and more importantly, the notification could be more effective. This example, although simple, shows the applicability of our model.

The process of selecting delivery candidates is NP-complete. However, we categorize candidates into public and private delivery and utilize the relationships between effectiveness and overtness, so that our algorithm can find appropriate devices and notification methods quickly in many cases.

Context is used extensively. In the filter stage, it is the input to atomic and complex modifiers, which refine the values of importance and willingness. In the delivery stage, it is the basis of feasibility criteria to eliminate infeasible delivery candidates. Both stages use context in a flexible way. The model does not depend on context to work, however, with more context, we can better refine the importance and willingness, and also produce a set of delivery candidates that is more feasible.

Figure 2.17 summarizes the intrusion problem from different views and the connections among them. This chapter focuses on the modeling of non-intrusive computing. It does not deal with middleware issues, but assumes the underlying platforms provide necessary support, such as context data and convenient accessing methods.

We conduct a user study to investigate how people choose different scales for parameters in the model. The results show that the answers of most participants are consistent and match the intuition. However, some may answer differently due to insufficient context and different interpretation of questions. The survey indicates that there may be some endowment effect, which could be handled using modifiers. Also, it shows that the effectiveness and overtness are not correlated as expected.

In order to evaluate the model, we need to study and analyze its performance. In the

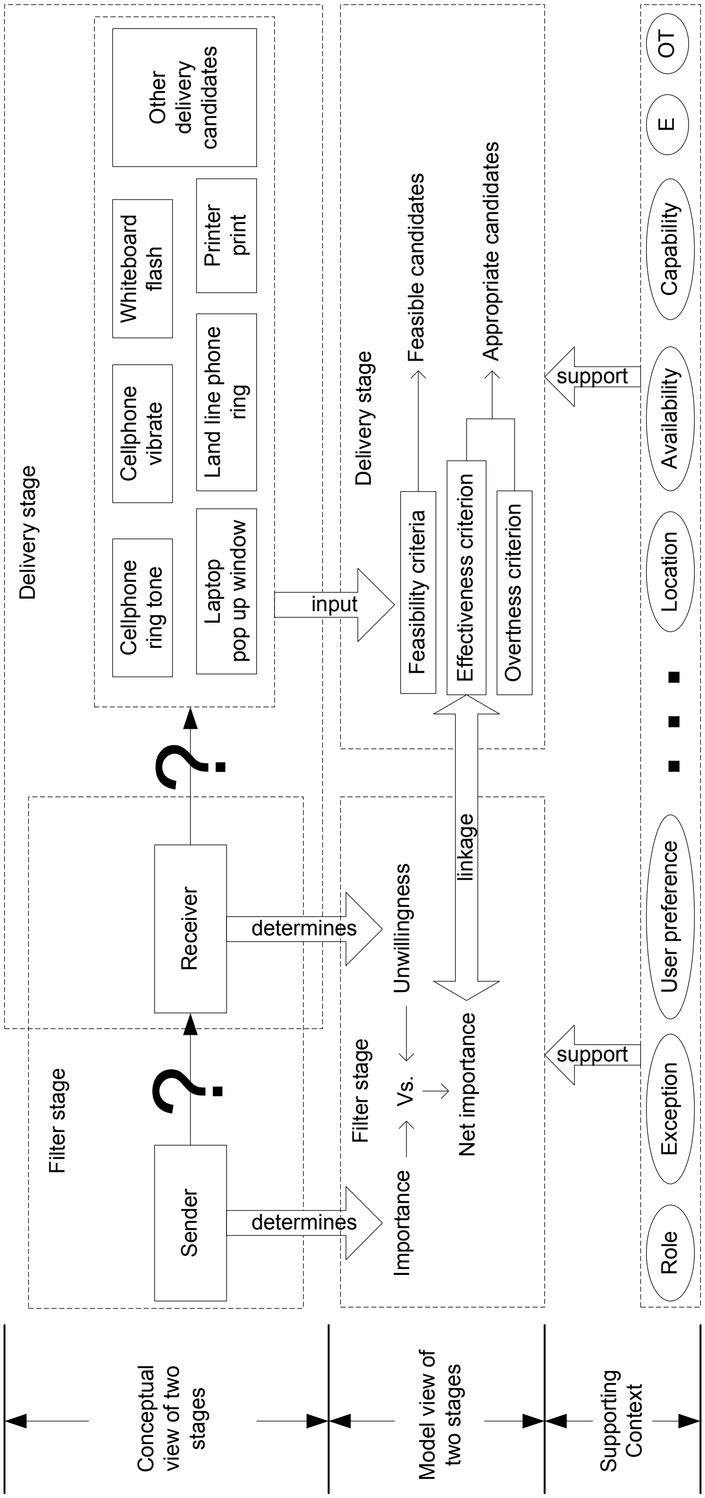


Figure 2.17: Different views of the intrusion problem

next chapter, we propose a performance metric and use several approaches to measure our model.

Chapter 3

Performance analysis

Non-intrusive computing aims to filter non-deliverable and hence intrusive interactions, and to choose proper devices and notification methods. Without this mechanism, trivial interactions might be delivered to cause intrusion to the receivers, or important ones may be missed due to the inappropriate selection of end devices. It is necessary to study and understand how interactions are handled normally without system help.

We call an interaction mis-handled if it is either non-deliverable but delivered or deliverable but does not reach the receiver. The interaction's deliverability is determined by the comparison model in Figure 2.2. The error rate ("errRate") measures the percentage of mis-handled interactions, as defined in Equation 3.1. "MisDelivered" and "misDropped" in the equation refer to these two cases respectively. The more a system considers intrusiveness, the less the error rate is. Hence, error rates of systems with little control on intrusiveness provide room for our non-intrusive-computing model to operate.

$$errRate = \frac{\text{misDelivered} + \text{misDropped}}{\text{total interactions}} \tag{3.1}$$

Next, considering a system with little support on controlling intrusiveness, we theoretically analyze its error rates from different aspects using different approaches. Some simulations are performed to confirm the analysis.

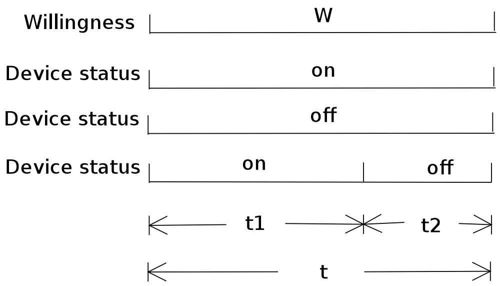


Figure 3.1: Constant willingness and different device-status patterns

3.1 Non-dropping approach

The first approach is called the non-dropping approach. In these scenarios, senders do not drop interactions; controlling non-deliverable interactions is solely on the receiver's side by setting the willingness and turning on/off the receiving device. In order to focus on analyzing the performance instead of worrying about the environment and its characteristics, we make the following assumptions.

- Incoming interactions arrive according to a Poisson process with importance distributed uniformly between 0 and 1.
- The receiver has only one device, which is able to handle all his interactions.
- The device has only two states, on and off. If it is turned on, it can receive any interaction, otherwise, it cannot, so the receiver misses all of them.

Figure 3.1 describes a scenario where the willingness is a constant W over time. There are three different patterns for the status of the device, always on, always off, and on for t_1 and off for t_2.

According to the definition of *errRate* in Equation 3.1, error rates for these three device-status patterns are $1-W$, W, and$((1-W)t_1+Wt_2)/t$ respectively. We can further deduce that, given a constant willingness, if the device is turned on and off more frequently, the error rate is the following (Equation 3.2):

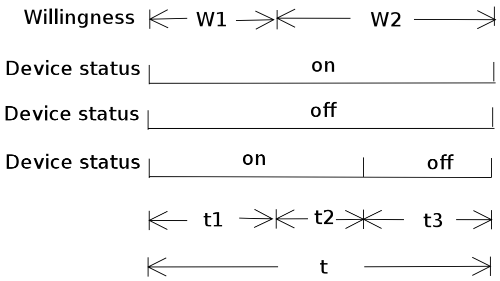


Figure 3.2: Different willingness and different device-status patterns

$$\begin{aligned} errRate &= \frac{(1-W)\sum t_i + W\sum t_j}{t} \qquad (3.2) \\ t_i &: \text{device is on} \\ t_j &: \text{device is off} \\ t &= \sum t_i + \sum t_j \end{aligned}$$

Now we consider the scenario where a user changes his willingness over time. Figure 3.2 describes such a scenario, where the willingness of a user changes from W_1 to W_2. The time spans for W_1 and W_2 are t_1 and $t_2 + t_3$ respectively. We use the same device-status patterns as those in Figure 3.1.

When the device is always on, the error rate is

$$errRate = \frac{(1-W_1)t_1 + (1-W_2)(t_2+t_3)}{t}$$

When it is always off, we have

$$errRate = \frac{W_1 t_1 + W_2(t_2+t_3)}{t}$$

When the device is on for $t_1 + t_2$ and off for t_3, the rate becomes

$$errRate = \frac{(1-W_1)t_1 + (1-W_2)t_2 + W_2 t_3}{t}$$

To generalize the calculation of the error rate, we divide the time t into small time spans t_i and $t_j^{'}$, where $t_i = 1,2,...,n$ and $t_j^{'} = 1,2,...,m$, such that within t_i, the device is on and the receiver's willingness is W_i; while within $t_j^{'}$, the device is off and the willingness of the receiver is $W_j^{'}$. Then the error rate is as follows.

$$errRate = \frac{\sum(1-W_i)t_i + \sum W_j^{'} t_j^{'}}{t} \tag{3.3}$$

In these receiver-controlled scenarios, error rates fluctuate and depend on the willingness and device-status patterns. It is difficult to draw an average. However, a few examples may help illustrating these rates.

Suppose Bob always leave his cell phone on, and his willingness is 0.5, which is constant over time. In this case, the error rate is 0.5. If his constant willingness is 0.2, the rate becomes 0.8. As another example, suppose Alice's willingness is 0.3 from 9AM to 12PM, and 0.7 from 12PM to 5PM. Her device is turned off before noon, and turned on after. Using Equation 3.3, we compute the error rate as 0.3.

With the non-dropping approach, if the receiver's willingness is consistent with the status of the device, e.g., if the willingness is low, then the device is off, and vice versa, the system shows a low error rate. This requires users to operate the device when they change their willingness. In reality, users may forget to do so, which tends to increase the error rate. The next section describes a different approach where senders control the interactions.

3.2 Dropping approach

With the previous approach, senders have no obligation for non-deliverable interactions, which are always sent. It is the receiver's responsibility to control them. Now, we discuss the dropping approach where the control is on the sender's side. An interaction may be dropped at the sender's discretion, as follows.

The sender estimates an unwillingness value for the receiver, V, such that if the importance of an interaction, I, is greater than V, then the sender processes it, otherwise, he drops it. However, the actual unwillingness of the receiver may not be the same as the sender's estimate, V. Hence, a delivered interaction may not be wanted by the receiver. For the same reason, a dropped one may actually be desired by the receiver. Note that we assume the importance, I, and the unwillingness, Unw, are independent, so are I and V. Other than this, we make similar assumptions to those of Section 3.1.

- Interactions arrive according to a Poisson process, with importance distributed uniformly between 0 and 1.

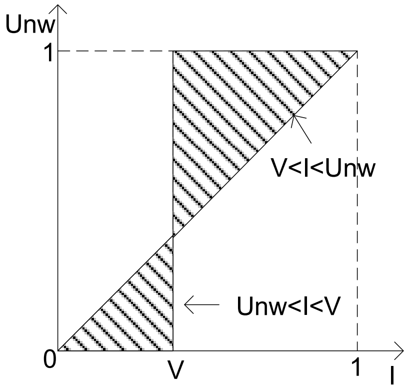


Figure 3.3: Calculating the error probability given a fixed V

- The actual willingness of the receiver is uniformly distributed between 0 and 1.
- The receiving device is always available, so it accepts any incoming interaction.

There are two situations where the sender may make inappropriate decisions. One is that an interaction's importance is greater than the estimated unwillingness but less than the actual one, in which case it is delivered and may cause intrusion to the receiver. The other is when an interaction's importance is less than the estimation but greater than the actual unwillingness, in which case, it is dropped but should have been sent to the receiver.

Let Unw be the actual unwillingness of the receiver. An interaction is mis-handled if $V < I < Unw$ or $V > I > Unw$, the probability of which, *errRate*, is calculated as the following, and also is illustrated by Figure 3.3.

$$\begin{aligned} errRate &= P(V < I < Unw \cup V > I > Unw) \\ &= P(V < I < Unw) + P(V > I > Unw) \\ &= \frac{1}{2}(1-V)^2 + \frac{1}{2}V^2 \\ &= \frac{1}{2} - V + V^2 \end{aligned}$$

The *errRate* is a quadratic function of V, which has the minimum value when the derivative is zero, i.e.,

$$\frac{d\,errRate}{dV} = -1 + 2V = 0$$

The minimum *errRate* is 1/4 when V is 1/2. This is to say that with the dropping approach, the best scenario is for the sender to assume the receiver's willingness is 0.5, which yields the minimum error rate, 0.25.

Note that in reality, sometimes people may have some knowledge about each other's current state and behave accordingly. For example, Alice knows that Bob is in a meeting every Friday morning, so she will not call him if it is not urgent. Hence 25% is the upper bound of the error rate using the dropping approach when the sender knows nothing about the receiver.

Now if we choose a random unwillingness for each interaction, assuming V is uniformly distributed between 0 and 1, the probability of an interaction being mis-handled becomes

$$\begin{aligned}
errRate &= P(V < I < Unw \cup V > I > Unw) \\
&= P(V < I < Unw) + P(V > I > Unw) \\
&= \frac{1}{3}(area_1 * height_1) + \frac{1}{3}(area_2 * height_2) \\
&= \frac{1}{3}(\frac{1}{2} * 1) + \frac{1}{3}(\frac{1}{2} * 1) \\
&= \frac{1}{3}
\end{aligned}$$

Figure 3.4 illustrates this probability calculation, which is the total volume of the two pyramids.

We have described two approaches, one is non-dropping, controlled by the sender, the other is dropping, controlled by the receiver. In reality, both might happen simultaneously. In the next section, we discuss a hybrid dropping approach, which combines the two.

3.3 Hybrid dropping approach

The non-dropping approach in Section 3.1 requires the receivers to turn on and off the device to control interactions, while the dropping one in Section 3.2 depends on the sender's estimate of the receiver's unwillingness. We now consider the strategies together by further assuming a probability of the receiver's device being off when the interaction

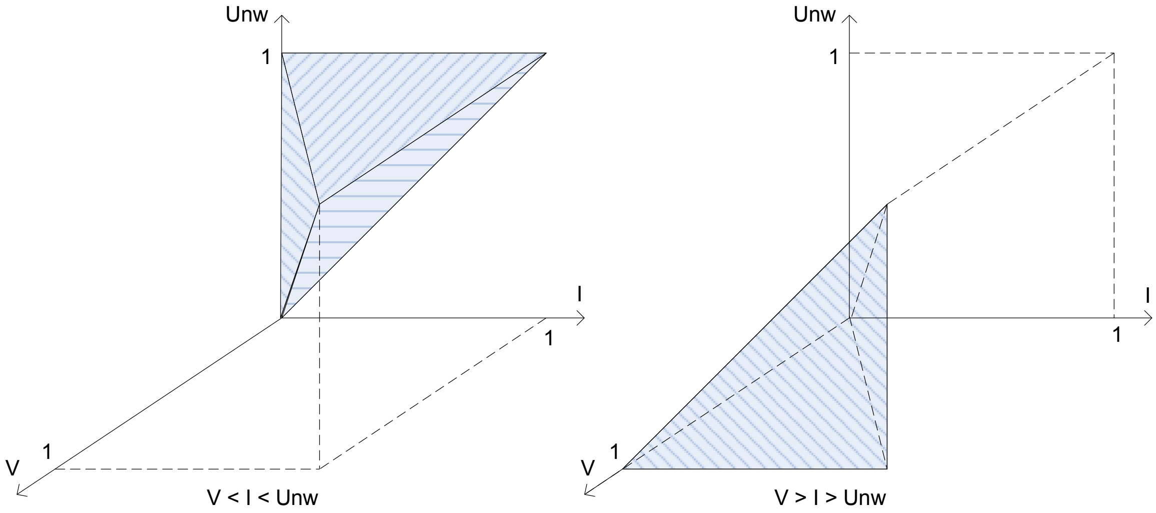


Figure 3.4: Calculating the error probability with random V

happens. Because the receiver might turn off the receiving device independent of the sender, the error rate is different from Equation 3.1.

Let I be the importance of the interaction, V the assumed unwillingness of the receiver, Unw the actual unwillingness, and P the probability of the receiving device being available.

When a sender decides to send an interaction based on the estimate V, i.e., $V < I$, it can either intrude on the receiver or not. Assume it is intrusive to the receiver, i.e., $V < I < Unw$. Now, if the receiving device has been turned off, then this intrusive interaction will not actually cause any intrusion to the receiver. Hence, the mis-delivered portion in Equation 3.1 becomes $misDelivered * P$. Similarly, if we assume the same interaction is not intrusive, i.e., $V < I \& Unw < I$, and if the device is currently not available, then the receiver will miss this deliverable and non-intrusive interaction, which is an error. If we call delivered non-intrusive interactions "deliveryAttempts," then the new portion of error is $deliveryAttempts * (1 - P)$.

On the contrary, if the sender decides to drop the interaction, i.e., $V > I$, the mis-dropped portion in the error rate will not change regardless of the status of the receiving device. Below is the revised error-rate definition reflecting these changes.

$$errRate = \frac{\text{misDelivered} * P + \text{deliveryAttempts} * (1 - P) + \text{misDropped}}{\text{total interactions}} \tag{3.4}$$

We conduct a simulation to show the result of the error rates with respect to the es-

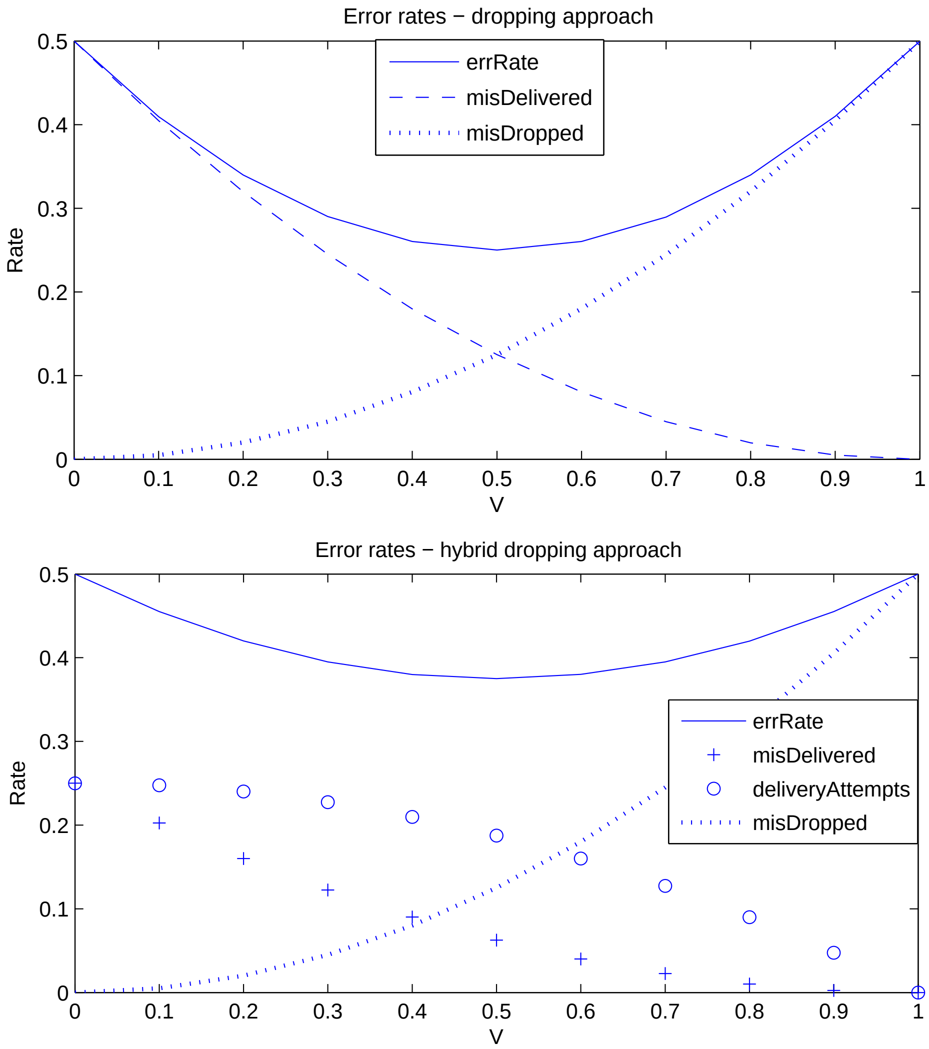


Figure 3.5: Error rates. Top: dropping approach; Bottom: hybrid dropping approach, $P = 0.5$

timate of the receiver's unwillingness, V, in Figure 3.5. There are two plots. The top one uses the dropping approach discussed in Section 3.2, while the bottom one uses the hybrid dropping approach with $P = 0.5$. Comparing these two, the mis-dropped rates are the same as explained before. The dropping approach produces lower overall error rates than the hybrid dropping one. When $V = 0.5$, both approaches perform the best, with error rates of 0.25 and 0.38 respectively. In detail, 0.38 is the sum of three rates, where $^{misDelivered*P}/_{total} = 0.0625$, $^{deliveryAttempts*(1-P)}/_{total} = 0.1875$, and $^{misDropped}/_{total} = 0.125$.

The summary and discussion of all three approaches is in the next section.

3.4 Discussion

We analyzed the error rates when people deal with interactions using three different approaches. The non-dropping approach is the simplest of the three. It assumes that senders have no responsibility to control interactions. The only control that receivers have is to turn off receiving devices to avoid intrusion. The side effect of this coarse mechanism is that they may miss important interactions. The error rates are highly dynamic and subject to whether receivers control devices to reflect their willingness in time.

The other two approaches, the dropping and the hybrid dropping ones, introduce an estimate of the receiver's unwillingness. The error rates are more stable and predictable as they use information from both the sender and the receiver.

We may do further analysis, such as by introducing the estimation error in the dropping-based approaches, or assuming a different distribution for incoming interactions, and so on. However, it cannot eliminate error rates. In fact, these error rates are due to senders and receivers taking independent action when interactions are taking place. On the one hand, the sender either takes no responsibility in sending non-deliverable interactions, or estimates the unwillingness before sending. However, due to lack of information about the receiver, the estimate may be different from the actual value, thus causing intrusion or incorrect cancellation to the receiver.

On the other hand, the receiver controls the device without knowing if interactions are about to happen. If he turns the device off, he may be able to avoid intrusive interactions but, at the same time, miss non-intrusive ones, and vice versa. Note that these analyses and discussions have not yet considered multiple devices and overtness to other people in the same environment as the receiver, which only increases the complexity and worsens the situation in practice.

Our non-intrusive model addresses these problems by effectively combining context from both the sender and receiver. Assuming accurate context is provided, our model can reduce the error rate from 38% (see Figure 3.5, bottom, $errRate = 0.38$ when $V = 0.5$) in the hybrid dropping scenario to zero. Furthermore, our definition of error rate does not include intrusion on other users, which will increase the rate if considered. That means our model can reduce the error from a higher rate than 38% to zero, because it considers overtness in selecting delivery candidates. An example calculation is the following.

According to the simulation in Section 3.3, the percentage of successfully delivered non-intrusive interactions is $deliveryAttempts * P = 0.1875$, where P, the probability of the device being available, is 50%. Suppose half of these interactions cause intrusion to other people in the same environment as the receiver, then the error rate becomes $0.38 + {}^{0.1875}/_{2} \approx 0.47$. Because our model addresses the intrusion problem on both the receiver and the other users, it can reduce the error rates from 47% to none, with proper context.

The error rates that we discussed in this chapter are the extent to which interactions can be handled better when there is a model and system in place than when there is not.

So far, we have described the modeling of non-intrusive computing in Chapters 2, and have analyzed its potential performance in terms of reducing mis-handled interactions. Since the model is built on top of a system and depends on system facilities, from the next chapter, we start to discuss issues at the system level, including a survey of previous pervasive-computing systems and our implementation of a non-intrusive-computing system.

Chapter 4

Previous work on pervasive-computing systems

This dissertation aims to address intrusion problems in pervasive computing. Our approach to this involves two layers, as explained in Section 1.2 (see Figure 1.2). Chapters 2 and 3 correspond to the upper modeling level, which concerns the model and analyzes its performance theoretically. From this chapter onwards, we focus on the bottom level, system construction. Before our non-intrusive-computing system is presented in Chapter 5, we review previous work on pervasive-computing systems. Section 4.1 is a literature review on selected systems. Sections 4.2 and 4.3 summarize them.

4.1 Literature review

Pervasive computing has been a current research topic for more than fifteen years. There are many systems and projects. We make no attempt to examine every one here. We also do not focus on specific aspects of pervasive computing, such as service discovery, user interfaces, and so on. Instead, we choose projects that are relatively comprehensive with many constructs and usually include some middleware platform for pervasive computing. These systems many be general purpose or more suitable to some environments such as domestic homes.

4.1.1 Aura

Although we briefly mention Aura in Section 2.1 with regard to the intrusiveness, we discuss it in more detail here from the system point of view. Aura [19] was an umbrella project at Carnegie Mellon University. It attempts to build a distraction-free pervasive-computing environment. The overall structure of Aura is shown in Figure 4.1, modified slightly from [19].

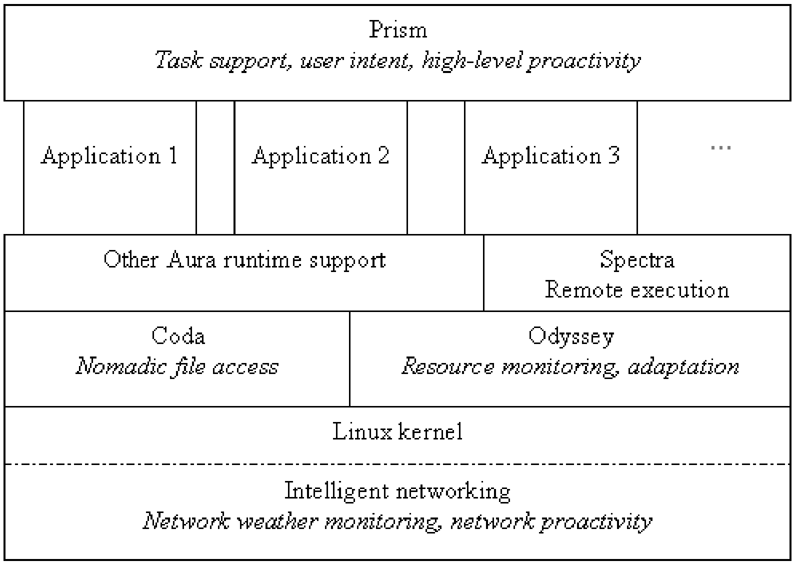

Prism
Task support, user intent, high-level proactivity
Application 1
Application 2
Application 3
...
Other Aura runtime support
Spectra
Remote execution
Coda
Nomadic file access
Odyssey
Resource monitoring, adaptation
Linux kernel
Intelligent networking
Network weather monitoring, network proactivity

Figure 4.1: Aura architecture

Aura is built on two other projects, Coda [46] and Odyssey [46]. Coda provides disconnected operation to support mobile and distributed computing. Odyssey features application-aware adaptation, which enables mobile applications to balance between resource consumption and fidelity. To be distraction-free, Aura encapsulates task-driven and high-level proactivity into another layer, Prism. Task-driven means that mobile users focus on the high-level tasks, and need not worry about lower-level configurations [54]. The system handles these for users in an automatic fashion. User intent is expressed explicitly in Prism, to guide the rest of the system, because user intent corresponds to the tasks. High-level proactivity means that when a user moves from one environment to another, the latter can obtain the execution states and necessary data from the former, and perform initialization automatically for the user. Figure 4.2, adapted from [19], illustrates the structure of Prism.

Prism is composed of a context observer, a task manager, an environment manager and many suppliers. The context observer is a general context manager. It collects con-

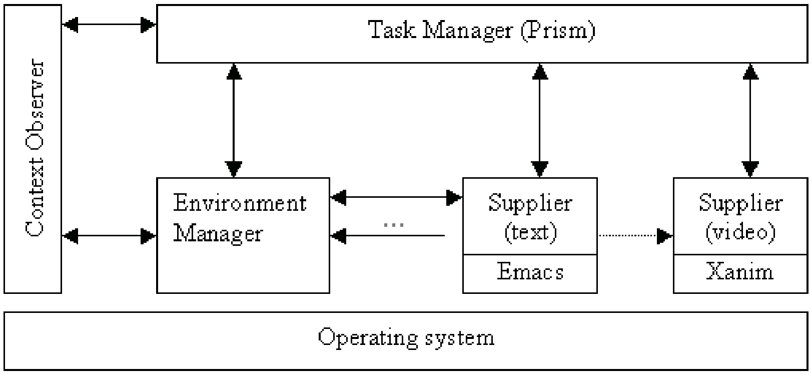


Figure 4.2: Prism architecture

text, detects changes, and reports relevant events to the task and environment managers. This context observer is also responsible for some simple authentication. When it detects that a user is approaching, it interacts with the user (usually through a handheld device) to perform authentication. There is no dedicated security-management component in the Aura/Prism architecture.

The task manager deals with user intent and maps it to system services. First, a user expresses his desires with explicit task descriptions, each of which is a coalition of services. The description uses abstract services, and hence is environment-independent. Then the task manager consults with the environment manager to map abstract services into actual ones in the current environment. Prism infers whether the user is going to stop one task or start another. In detail, it is the context observer that handles this simple prediction based on the context and user-specific information, such as the calendar.

The environment manager registers the available services and devices, and is responsible for mapping abstract services into actual ones based on descriptions. The system is adaptable in that when a service becomes unavailable at runtime, the manager tries to find a replacement.

Suppliers represent individual services with abstract descriptions, so that the environment manager can translate the desired service, described in an abstract format, to a current instance. Note that Aura assumes that both users' and suppliers' abstract descriptions of services use the same vocabulary.

Aura does have some runtime support. The environment manager stores some of the task-related data, such as task-execution status. The majority of the data is stored in the Aura file system, which is not included in Prism.

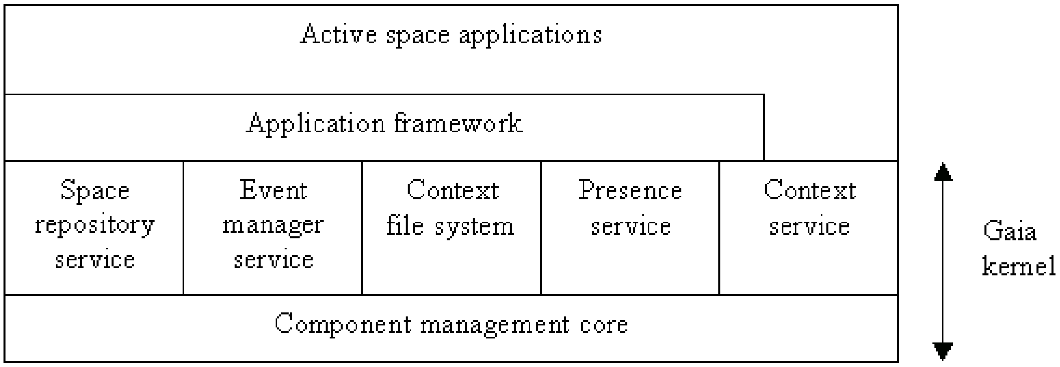


Figure 4.3: Gaia architecture

Aura uses an environment-independent format to describe tasks explicitly, and these tasks are initialized automatically in different environments. Thus, a task can take advantage of different resources in different environments, freeing the user from configuration management. The Aura project seems dormant based on its website (`http://www.cs.cmu.edu/~aura/`). The latest work was in 2004.

4.1.2 Gaia

Gaia is also mentioned when we discuss previous work on intrusiveness in Section 2.1. We now describe it because it is a comprehensive system and provides middleware for ubiquitous computing. Gaia [43] was developed at the University of Illinois at Urbana-Champaign. Gaia attempts to provide users' data wherever they are. In Gaia's view, applications usually include several devices during execution, and applications should be described using abstract generic devices. The runtime system maps abstract devices to concrete ones depending on the environment. The architecture of Gaia is in Figure 4.3 from [43].

Gaia uses the term "active space" to denote a smart space. The Gaia kernel consists of several components: context service, presence service, context file system, event-manager service, and space-repository service. Above the kernel, there is an application framework that uses the model-view-controller (MVC) architecture for application development, control, and presentation.

The context service obtains context from sensors, and manages a registry. Any application or component can register interest in the context service. The context service uses first-order logic and boolean algebra to infer high-level context.

The presence service manages services and devices. It reports which services are available in the current system, which applications are running, who uses what device, and where the users are. Simply put, the presence service observes all the software and hardware resources, and the space repository maintains a complete list of resources (hardware and software entities) in the environment.

The space repository is similar to a naming service. It stores attribute information such as name, type, and owner for each software and hardware component, and provides a query mechanism for applications. The difference between the space repository and the presence service is that the latter keeps a live resource list, while the space-repository service stores the attributes of the various resources. Both services manage resources, but at different levels. When the application is initialized in a particular environment, it queries the space repository and maps the abstract resources (software and hardware) into actual ones by specifying attributes.

Event service is the communication mechanism among components in Gaia. It manages events that change the state of some devices or software entities. It separates event suppliers and consumers and creates a channel for each kind of events.

The context file system (CFS) is a traditional file system augmented with context information. In the CFS, any file or directory can be associated with certain context, such as time, location, topic, or number of attendees in a meeting. The CFS constructs a virtual file system based on context, i.e., it collects files located at different machines but with the same context into the same directory in the virtual file system. Thus, applications can obtain the desired data by specifying context. CFS can also make user data from other smart spaces available automatically. At each smart space, there is a mounting server that can mount files or directories from other smart spaces. The user's handheld will carry the mount points, which give the locations of the personal and related data. When this user enters a space, the mounting server retrieves mounting-point information from the handheld and mounts files automatically.

Gaia supports both traditional desktop computers and mobile handhelds when performing the device mapping. In order for an application to use a mobile device, Gaia introduces a microserver [12], which is a component running on the mobile device that helps export the device's native environment and functionalities to the Gaia system, so that Gaia applications can use them. Figure 4.4 is an illustration, modified from the one in [12]). A Gaia application sends a request to the Gaia proxy that hides the Gaia specifics from applications. The middleware component on the mobile device is a J2ME compo-

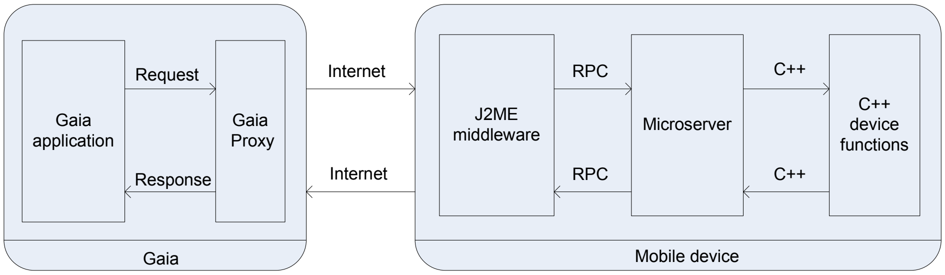


Figure 4.4: Call sequence using Gaia microserver

nent providing a Java sandbox for the device. The microserver handles the device's native environment and export it to the J2ME middleware that further interprets this environment into a number of Gaia services that applications can use. The microserver breaks the limitation confined by the standardized set of functionalities that J2ME allows and enables the integration of arbitrary native devices.

Overall, Gaia is a pervasive-computing environment where applications are developed in terms of abstract devices. At runtime, these devices are instantiated to detailed, and potentially different ones in each smart space. The context file system keeps users' data available regardless of location. Gaia uses ontologies and first-order logic for representing and reasoning about context. The Gaia microserver incorporates mobile devices into active spaces and Gaia applications may use the native functionalities on these devices. The Gaia project started in 2000, and seems to be dormant after 2005 according to its website (`http://gaia.cs.uiuc.edu`).

4.1.3 Stanford interactive workspaces project - iRoom

The interactive workspaces project [30, 31] at Stanford University was started in 1999. It focuses on the collaboration of devices and applications in a pervasive-computing space. iRoom is an experimental research environment, rich in devices and technologies. This project has three goals. One is data moving, which means data can move freely between applications running on different devices. Data formats are converted automatically to support a specific device. The second goal is control moving, by which users can control the same application on different devices using different interfaces. These different interfaces are generated automatically by the system. The last goal is dynamic application coordination, which means that in an interactive workspace with many applications coupled together, when events occur in one application, related ones will be aware of these

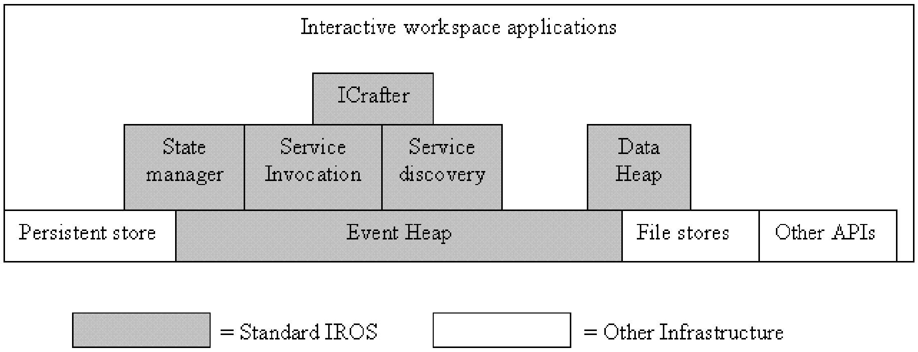


Figure 4.5: iROS

events and make corresponding changes.

To enable these goals, Stanford researchers developed a meta-operating system, the Interactive Room Operating System (iROS), as shown in Figure 4.5 [30].

The three main components in iROS are the Event Heap, the Data Heap, and the ICrafter. The event heap is a collaboration framework. Events in iROS are represented by tuples, which are attribute-value combinations. The event heap is based on the tuple-space-collaboration model that is temporally and referentially decoupled. Applications can retrieve events using pattern matching, i.e., specifying some fields of a desired tuple, and then searching the tuple space. The event heap extends the regular tuple space by adding an important feature, tuple expiration. It associates each tuple with an expiration date to prevent event tuples from staying in the tuple space forever. The event heap also incorporates some other features, such as tuple sequencing and tuple routing [29].

The event heap is vital for dynamic application coordination. It provides a set of APIs for applications to operate on event tuples, and applications can communicate through the event heap. Because the event heap is temporally and referentially decoupled, communication is asynchronous and applications do not need to specify receivers when generating a tuple.

The data heap stores data for applications. Data has attributes. Applications locate data by attributes without knowing its physical location. The data heap can also invoke format converters automatically to transform data from one format to another. Hence, data can be communicated among different applications.

The ICrafter [40] is a service framework. It is responsible for service invocation and dynamic interface generation, and is based on the event heap. When a user wants to invoke a service, he sends a request event to the heap. An interface manager, a sub-component inside ICrafter, retrieves the request and searches the heap for appropriate service descriptions. Then it generates an appropriate interface for the requesting device and places the interface in the event heap. Next, the user can retrieve the interface and invoke the service through the heap.

ICrafter has an intelligent function, service aggregation. For example, if a user wants to take a picture and then print it, she needs to have a camera service as well as a printing service. Intuitively, she needs to obtain the camera service first, take a picture, save it somewhere, ask for a printing service, and finally send the picture to the printing service. Obviously, these two services are related and should be aggregated. ICrafter can return to the user an integrated interface to these two services, which will help her take the picture and send it to a printer. ICrafter uses patterns to implement service aggregation. For example, the picture-printing example belongs to the consumer-producer pattern, as the camera is a producer and the printer is a consumer.

The interactive workspace project does not concentrate on context sensing; instead, it focuses on the collaboration of applications and builds a framework for users to move data and control applications, and for applications to collaborate with each other in a technology-rich environment. The iROS system uses a central server to host the event heap, which is an extended tuple space and vital to the rest of the system. The advantage of this is that communicating applications do not need to know each other beforehand, and two parties are not required to be present at the same time to make the communication happen. The last active update on this project was in 2004.

4.1.4 *One.world*

The *one.world* [21, 22] project was developed originally at the University of Washington. This project provides a framework for pervasive-application development and a platform for pervasive-application execution. If each device has the *one.world* system installed, then applications can move from one device to another. Tuples and asynchronous events are used as the data and the collaboration models (see Figure 4.6 [21]).

One.world aims to address three problems in pervasive computing, which are constant change of context, ad-hoc composition, and pervasive sharing. In a pervasive-computing environment, context changes often and failures are inevitable, and the system should expose this information to applications, so that the applications can adapt to changes, rather than forcing that on the users. Ad-hoc composition means that when

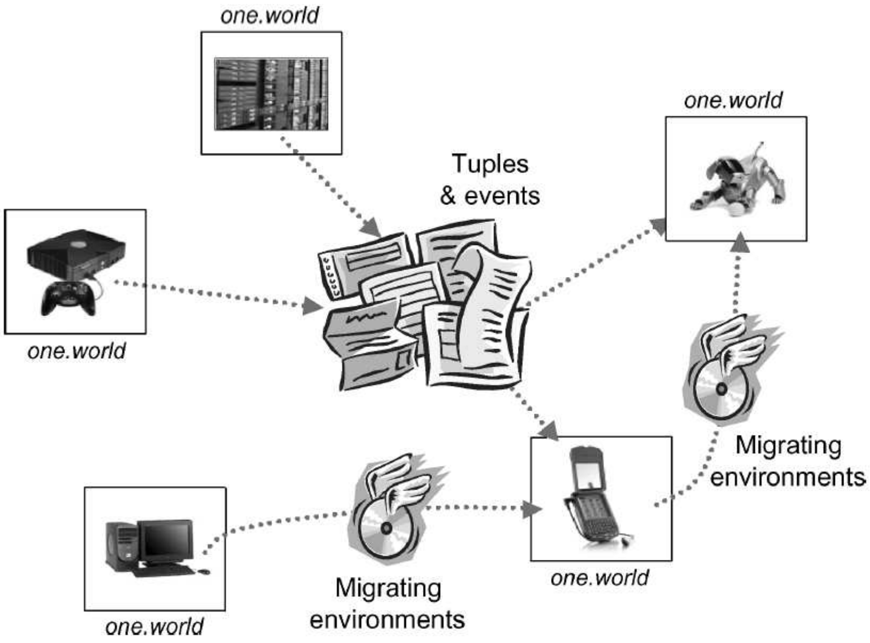


Figure 4.6: *One.world* illustration

users move to a different place, devices and applications can just “plug together” in the new environment. Pervasive sharing means that information should be accessible anywhere and anytime.

To address these problems, *one.world* provides corresponding services in three layers, foundation services, system services, and user space. The architecture of *one.world* is shown in Figure 4.7 [21].

Foundation services are those that address contextual change, ad-hoc composition, and pervasive sharing. The virtual machine is the underlying execution support, such as a Java virtual machine, or Microsoft common language runtime. Tuples are attribute-value pairs, which is claimed to facilitate data sharing in *one.world*. The asynchronous-events module is the communication module, where contextual changes can be represented by events. The environments module holds the code, data, and other, if any, environmental information for the application. Each application corresponds to an environment. Environments can be nested, which is used for application composition.

System services include migration, checkpointing, discovery, remote events, the tuple storage, and the query engine. The checkpointing service captures application-execution states in an environment for later restoration or migration. The migration service moves environments from one device to another by moving the states captured by checkpoint-

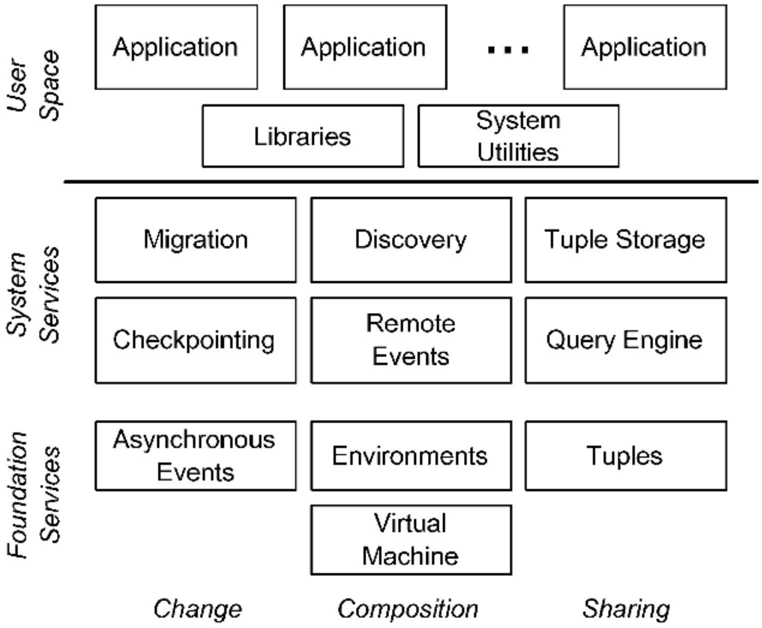


Figure 4.7: *One.world* architecture

ing. The discovery service can find other one.world-enabled resources. The remote-events service provides a way to deliver events to remote applications, which is the basic communication mechanism in *one.world*.

Above system services, there is the user space. Libraries and system utilities that might be useful for user applications are defined here. For example, the user-interface library is in this space. The user space also holds user applications. Experimental applications and services are built on *one.world*, such as the user and application manager, and the text-and-audio-messaging system [22].

Overall, *one.world* provides a framework for building pervasive applications. The *one.world* system should be installed on attending devices before data and applications can move around. From the user and the developer point of view, it provides a single consistent computing and communication platform. The system is designed from the ground up to solve the requirement for pervasive-computing-application development. However, a limitation in *one.world* [22] is that it uses a programmatic data model, i.e., using Java classes for tuples. In order to process a tuple, the corresponding Java class has to be available. A better solution is to use a data-centric data model such as XML Schema to represent tuples, which makes them easier to distribute and share than Java classes. The last report on the *one.world* project was in 2004.

4.1.5 Plan B

Plan B is an operating system for ubiquitous-computing environments developed at the Universidad Rey Juan Carlos, Spain [9, 8]. It is derived from Plan 9 [39], which is a distributed operating system that uses Unix files to represent specific interfaces, such networking and user interfaces, and was developed by Bell Labs in the 1990's.

Plan B follows the same ideas as its predecessor and extends the system to pervasive computing. Popular elements in such environments, such as sensors, actuators, context, events, services, and so on, are all represented as files and are handled using file-system operations. Plan B uses a network-file-system protocol from Plan 9 to connect different machines and import/export file systems.

Unlike other pervasive-computing systems and projects, which usually employ middleware to provide high-level abstractions and hide the heterogeneities in the environment, Plan B does not have this notion and instead uses files universally. Hence any device capable of importing remote files and exporting local files can be part of a Plan B system; applications become compositions of file operations, such as mount, open, read, write, close, and so on. Users can use any languages or tools that can use files, such as the shell and C, for the programming.

In a Plan B system, there are multiple machines, each of which has a number of resources, such as devices attached, programs and services that are running, the owner of the machine, and so on. When a machine boots, it exports all its resources to the network. Each resource is exported as a file system, called *a resource volume*. Each volume has a set of attributes, called *constraints*. These files systems may be virtual, similar to */proc* in Unix systems. Figure 4.8 [8] illustrates an example of a Plan B computing environment. Machine #1 exports four resources, i.e., a mouse, a keyboard, a screen, and one or more X10 devices; machine #2 exports two, i.e., an "exec" allowing users to launch applications and a home directory. According to [8], a *kbd* volume is a virtual file system with a single directory containing two files: *cons* and *kbdctl*. Reading from *cons* receives key strokes that a user enters; writing to *kbdctl* sends control information, such as redirecting the keyboard to another GUI. The environment of each application consists of resources imported by the application. Each user's environment includes all applications' environments that belong to him.

Machines are arranged using a peer-to-peer architecture, because each machine runs the same Plan B system (see Figure 4.9 [9]) and can import remote resource volumes

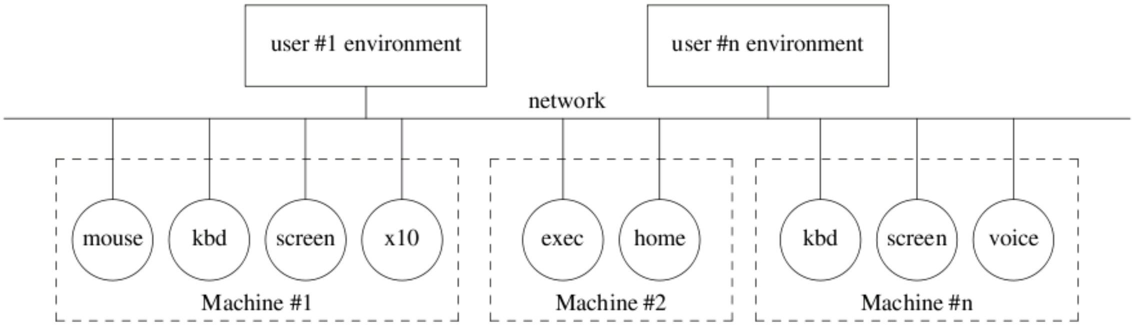


Figure 4.8: A Plan B computing environment

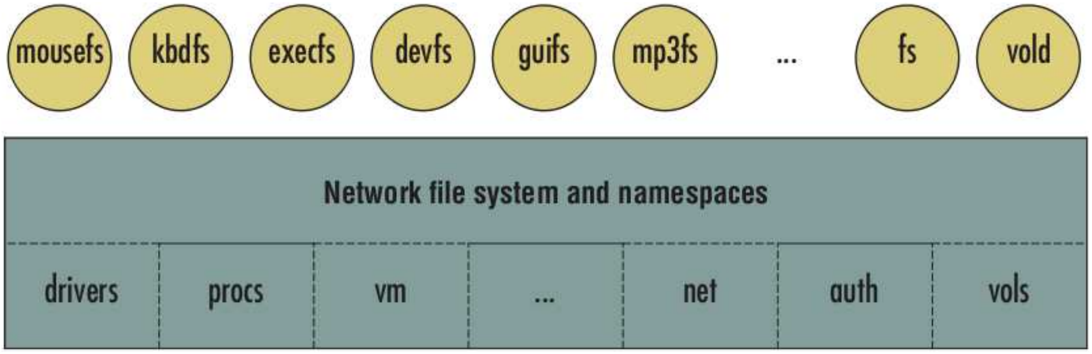


Figure 4.9: Plan B architecture

as well export local ones. A Plan B system contains kernel modules and user modules, represented by squares and circles respectively in Figure 4.9. Among those modules, *vols* manages all resource volumes that the kernel knows, and *vold* discovers volumes across the network. More information about these modules can be found in [9, 8]. Note that not all machines have to run Plan B. As long as a machine can mount Plan B file systems, it can use services provided by those files. Similarly, a system can export some resources without installing the complete Plan B system. These two features facilitate integration with other systems.

Context in Plan B is organized with three types of volumes, i.e., */who*, */where*, and */what* for users, places, and things respectively. Similar to other volumes, they contain directories and files to hold information. These volumes form a hierarchical structure to represent context. For example, */who/user/where* represents the last known location of a user. As another example, */who/user/status* reports the status of a user in a way similar to instant-messaging systems. To use context, users and applications read context volume files. For example, *ls /what* displays all machines in the current environment. To handle the dynamicity and context change, Plan B can adapt applications to new or available resources based on users' specifications and the constraints of volumes. In Plan B, permanent context is kept in real files, while temporary context is kept in virtual files in RAM. Users can create new context by creating new files in the corresponding volumes.

To interact with devices, users or applications read or write the resource volumes of these devices. Plan B has an event-delivery mechanism, implemented using volume *portfs*. Port files may be created dynamically in *portfs*. Each message written to a port file is delivered to all applications and users who read the same file.

To summarize, Plan B takes a different approach to pervasive computing. It abstracts the environments using files and file-system operations, which are well established. The networked computing platform is transparent to users and applications as if it were a single operating system. Plan B is also aware of context and adaptable to context and environment changes. However, as pointed out by [9], Plan B lacks a type-checking mechanism. The control information is in the form of text. Upon erroneous operations, the system will have to wait for user intervention to correct them. Because Plan B is built on the operating-system level, it is difficult to interoperate with non-Plan-B systems, such as Windows. Also, using file operations to interact with the environment gains simplicity, but at the same time, is less intuitive and may be inconvenient for the end users.

4.1.6 Cooltown

Cooltown [38] was a pervasive-computing project initiated at Hewlett-Packard around 2001. Its goal is to build a web-based nomadic computing system. Web presence is the central idea of this project, which means that people, places, and daily objects all have web representations [33]. Simply put, it applies the web-page concept to people, places, and things. Cooltown proposes that a nomadic user can get any desired information if there is a web presence for everything, which bridges the difference between physical space and cyber space. The overall architecture of this computing system is shown in Figure 4.10, taken from [32].

In Cooltown, each physical object, such as a picture or a printer, is associated with some web page that describes the object. The URL of the page is stored in some beacon, tag, or explicit text, which is shown at the bottom of the Cooltown architecture. Users have a handheld with a specific reader installed that can fetch the URL information from scanning a tag, sensing a beacon, or just reading a text description. This is called URL

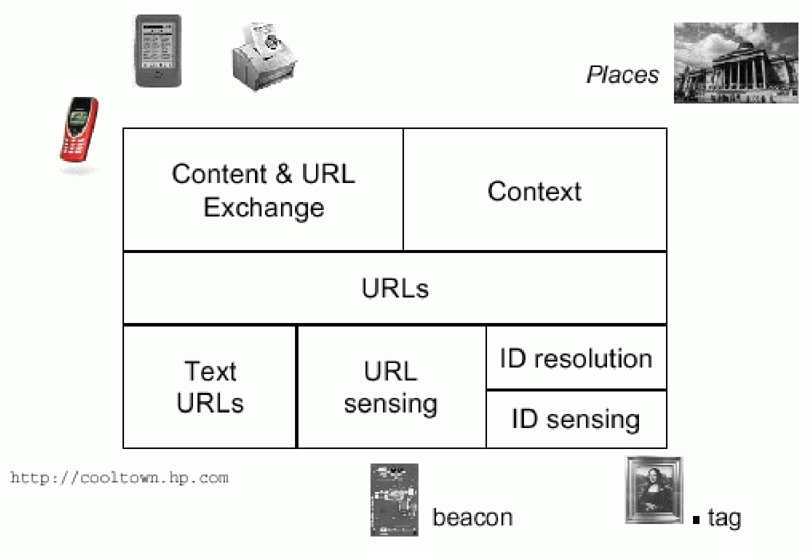


Figure 4.10: Cooltown

sensing or ID sensing. After that, the user can exchange content with different objects. For example, in an exhibition hall, a user finds the URL of a painting and a printer. He reads the page at the URL, which describes the picture in detail. He realizes that he really likes the painting. Then he checks the page of the printer and determines that it is a color printer with high resolution. He sends the URL of the painting to the printer, which then fetches the image and prints it. The user obtains a printout of his favorite painting when he leaves.

The "place" is the primary notion of context in Cooltown. It is a physical container that includes things and persons. The place itself also has a web presence, which describes the place and things and persons in it by adding their URLs to its web page. By checking the place's web presence or web page, the user can receive information about current things and persons it contains.

Above the web presence, illustrated by "URL" in Figure 4.10, Cooltown concentrates on content exchange among objects, especially different devices. There are three ways of doing that. First, one device can send content directly to another. Second, a device can send the URL of the content to another, and the second device can retrieve the actual content through the received URL. Third, a device can expose a configuration interface on its web page. Other devices can configure it before use. For example, a printer supports several resolutions; other devices can set the desired resolution through the printer's web page. So in this way, the initiating device can read the state and supported configuration parameters of the destination device, and send content and configuration parameters together to it, so that the content is handled in a personalized fashion.

Cooltown simply relies on the web. It does not require the handheld devices to have specific software, such as a Java virtual machine. It provides service management and device management through the web presence of each place. Each device provides an operation interface through its own web presence, so that as long as a user obtains the URL, he can control the device. It does not require a centralized infrastructure, except that the web presence of a place maintains the available resources in it centrally.

From the middleware point of view, Cooltown does not provide many services for pervasive computing. However, Cooltown provides a familiar interface to users, i.e., the web interface, through which users, environments, entities are integrated. Cooltown uses a peer-to-peer architecture, while most of the similar systems use a centralized one. The Cooltown project is dormant according to HP's website (`http://www.hpl.hp.com/techreports/2001/HPL-2001-22.html`).

4.1.7 The Aware Home

The Aware Home [2] at the Georgia Institute of Technology investigates health-care issues of pervasive computing in a real house. The purpose of this project is to explore how computing technologies can help people, usually elders with cognitive disabilities. Before we describe the project, we discuss some general characteristics of health care as an application area for pervasive computing.

First of all, the intent in applying pervasive computing to health care is to help people live healthier lives, especially older people. Traditionally, elder care occurs in nursing homes and hospitals, with significant reliance on professional caregivers. We expect pervasive-computing technologies might help provide a home environment with almost the same level of health care as a hospital. This is the long-term goal in applying pervasive computing in health care.

Second, there are several categories of users, such as patients, healthy family members, caregiving nurses, and doctors. Family members could be caregivers and supplement professional nurses. It is important to distinguish these categories, because the pervasive-computing system should react differently to different kinds of users, or even to the same kind of users but doing different jobs. For example, a healthy family member goes to the bathroom to open the shower tap, and then the system should decide if it is the family member who wants to take a shower or if he prepares a shower for the patient, so the system can adjust the temperature of the water. A pervasive-computing system for health care should provide facilities for all of the users in the environment, and this system needs to be careful if one user's behavior affects others. For example, when a family member wants to play rock music, although this behavior is appropriate for healthy members, it might be inappropriate for the others residing in the same room. Generally speaking, for the residents, the system should help them avoid dangers, such as making sure that they take medicine only at the appropriate times, or suggest they avoid dangerous situations. For the caregivers and doctors, the system should help them observe the behaviors of the residents, and provide history data for monitoring and diagnosis.

The third characteristic of pervasive computing in health care is the extensive need for context. The system should know exactly what is happening at all times. This relies heavily on sensors and cameras, which can locate users and record their behavior. Users may also have wearable computers and electronic badges, to cooperate with the system.

The last characteristic is that the patients or the elder residents may not always have a clear intent about what they want or what they want to do. As a disease such as Alzheimer's progresses, their intent may even be inappropriate. A system should be able to detect and infer residents' intent. If it is appropriate, such as a user vaguely remembering to take medicine before dining, the system should assist users, such as by reminding them where the medicine is and so on. Otherwise, the system should notify the caregivers and intervene necessarily if users intend to perform actions that may hurt themselves. This does not mean that the system should override users. Instead, it should be careful and behave in a way that makes users feel comfortable. In general, the system should remain backstage and assist users unobtrusively, not disturb them.

Although there are some special considerations in health-care applications, the theme of pervasive computing does not change, which is still the seamless integration of all kinds of devices and services, calmly and invisibly. Now we return to the aware-home project.

This project has been deployed in a real house since 1999. The project has three main goals. One is crisis prevention, which tries to prevent bad things from happening to elder people. The second is memory augmentation, which attempts to compensate for a lack of short-term memory. This happens especially in situations that involve many steps, such as cooking. Memory augmentation is implemented through many digital cameras, which

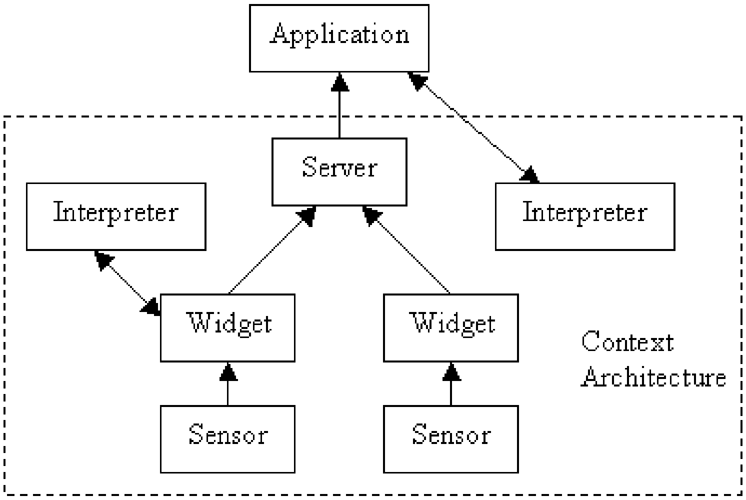


Figure 4.11: Context-based infrastructure of the Aware Home

take live pictures at some frequency. When a user forgets what he did, the system can play back these pictures to remind the user. The third goal is social connection, especially supporting the connection between elder people and their adult children. Adult children do not usually live with their parents, so they often worry about them. Thus, the project seeks to build a communication channel between parents and children, which will give both live pictures of each other. This provides "peace of mind."

Obviously, context-awareness is the core of the Aware Home, since the system needs to know the house in detail. Many sensors are installed in the home, such as the smart floor. Each sensor is represented by a software widget, which has attributes, values, and associated events. A context server manages widgets in terms of aggregation of context. When dealing with context, the Aware Home also has some interpreters to abstract raw context into meaningful information. Applications can obtain interpreted and/or aggregated context information from the context server. This forms a context-based infrastructure (see Figure 4.11 from [16]) for the Aware Home.

The overall system architecture of the Aware Home is illustrated in Figure 4.12. We can see that INCA & GRBAC are based on the context infrastructure. INCA stands for infrastructure for capture and access. It provides architectural support for applications that need automatic capture of and access to data, including storing history data for future access. In a health-care pervasive-computing environment, historical data is very important to both the system and the caregiver, so that they can track the behaviors of older people.

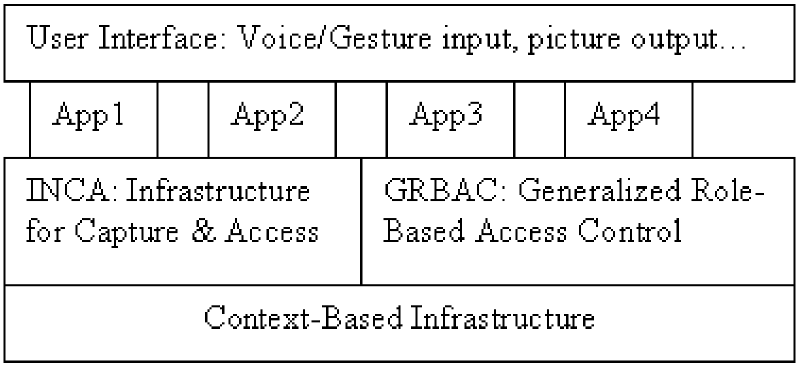

User Interface: Voice/Gesture input, picture output...
App1
App2
App3
App4
INCA: Infrastructure for Capture & Access
GRBAC: Generalized Role-Based Access Control
Context-Based Infrastructure

Figure 4.12: The Aware Home architecture

INCA is implemented as a generic database system with some common services. It is generic in that it supports any kind of data. Common services are services that may be useful for many applications, such as audio capture, and attribute-based access and information integration.

GRBAC is an extension to traditional role-based security systems. Role-based access control (RBAC) defines a users' access based on the user role. For example, if a user has the role "parent," he has the right to do everything at home. Otherwise, if a user has role "child," he can only do more limited things at home. GRBAC extends RBAC by introducing object roles and environment roles. Environment roles define where or when things happen. Object roles define what kinds of objects are manipulated. For example, we can define "weekend" as an environment role, and define "TV," "VCR," and "game box" as object roles: entertainment devices. Then we can say children (a role in RBAC) can use entertainment devices (object role in GRBAC) on the weekend (environment role in GRBAC). GRBAC implements security management in the Aware Home.

At the layer above INCA and GRBAC, there are applications designed to support older people in a pervasive-computing environment. Examples are the digital family portrait [37] and the personal audio loop [25]. The digital family portrait is an application that connects two houses; at one house, digital cameras take live pictures and the application sends pictures to another. In this way, children at a remote house can know the current situation of their parents. They can verify that their parents are well. The personal audio loop is an application that captures audio at one place and plays that audio back elsewhere when required.

The Aware Home research involves computer science, medical science, gerontology, and other areas and is now growing to a large research base where many interdisciplinary projects are developed actively. The interdisciplinarity of the Aware Home also exemplifies a recent trend in pervasive computing, which is exploring it in other domains, such as health care, transportation, sports, real-world deployment, urban computing, and so on. These fields provide many new opportunities for pervasive computing. However, the progress and success of interdisciplinary research depends on pervasive-computing middleware, platform, and integration, which is the focus of this literature review.

There are too many pervasive-computing systems to be listed. We select and discuss those that are relatively large and act more as computing platforms than applications. The next two sections summarize them by discussing their recurring themes and insufficiencies.

4.2 Recurring themes

Pervasive computing is a trend that emerged from distributed and mobile computing. Its goal is to integrate devices and technologies in smart spaces so that they weave themselves into people's daily lives. Many projects have detailed this vision from different perspectives, such as adaptive systems, context-aware computing, data and program migration, and so on. To achieve these aspects, abstractions are needed, such as the abstract task description in Aura or using files for resources in Plan B. Also, although these abstractions may be at different levels, like the middleware level or the operating-system level, they all provide users with a consistent view of the system, so that users and developers concentrate on high-level logic, while the platforms handle the environments. Although different systems have different designs and goals, they exhibit some similar features that we call recurring themes. Figure 4.13 illustrates a general architecture of a pervasive-computing platform. Each module represents a recurring theme.

At the bottom level, there is the runtime and communication module. Above this, there are five fundamental modules for security, context, services, devices, and data. The user interface is built on top of these basic modules. Then there are the users and applications. Note that the boundaries of these modules are not always clear and precise, and the layering may be different. For example, in some sensing-intensive systems, devices, such as sensors, are usually at the bottom of the system. Hence, recurring themes are discussed more from the functional than structural point of view.

The runtime module provides communication and coordination support, such as re-

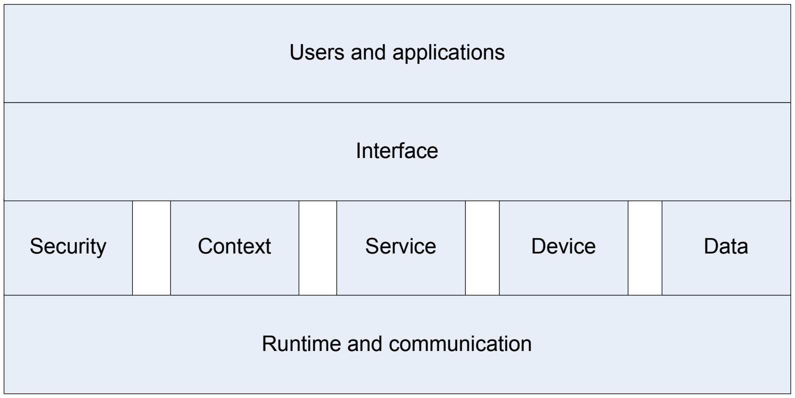


Figure 4.13: General architecture and recurring themes

mote-method invocation and remote-procedure call (RMI/RPC), message-based communication, stream-based communication, publish-subscribe-based communication (pub-sub), or tuple-space facilities. The runtime may also be responsible for naming in the system.

The security module is responsible for authentication and registration, access control, and secure transmission of information. Users in a pervasive-computing system are nomadic. Some of them are static, such as employees; some are dynamic, such as visitors. The security module should grant different permissions to different kinds of users, instead of simply disabling visitors. A pervasive-computing system tends to have many more devices and resources than before. A flexible access-control strategy over users accessing resources is necessary. Also, wireless network technologies are used extensively nowadays. The security module enables secure transmission of information over both wireless and wired networks.

The context module concerns context representation, context collection, context interpretation (processing raw context into a form understandable to a system or program), and context composition (synthesizing several low-level contexts into a high-level one). Context information is usually collected by various sensors and is hence heterogeneous. Some is continuous, such as room temperature; some is discrete, such as a telephone ring; some is static, such as a room number, and some is dynamic, such as the number of users in the room. An important function of this module is masking the heterogeneity of context and providing the rest of the system with a universal interface to deal with context. Accessing context may also be part of a context manager. Many systems provide a publish-subscribe mechanism to facilitate accessing context. A typical context manager

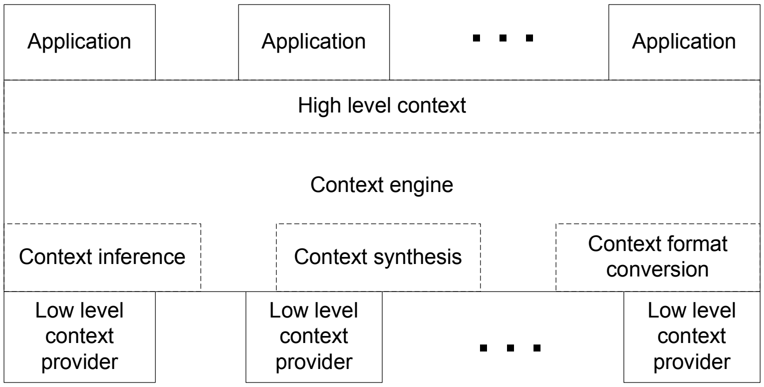

Application
Application
Application
High level context
Context engine
Context inference
Context synthesis
Context format conversion
Low level context provider
Low level context provider
Low level context provider

Figure 4.14: Context manager

is shown in Figure 4.14.

The service module mainly concerns service registration and discovery. Service providers first register themselves with this module, then applications discover services dynamically. Different vendors could provide services in different formats. This results in heterogeneous services. A pervasive-computing system can either use some wrapper for services, or require services to follow a set of standards. In addition, applications and users are dynamic, as they come and go. Users walking into a new environment cannot necessarily know the available services in advance. So they tend to describe services abstractly. The service module thus should be able to find the appropriate actual services to match the abstract descriptions.

The device module keeps track of devices in the current environment. Devices may be sensors and actuators; they may also provide some services, such as a printing service provided by a printer, or use other services, such as a handheld using a desktop computer to compute some tasks. The device module and service module are overlapping to some extent. However, many existing systems tend to have a dedicated device manager to make the system flexible.

The data module concerns information about or related to the user. This could be work documents, personal preferences, application-execution status, entertainment files, address books, or any other personal data. This module may perform two main functions.

First, it keeps track of the data and synchronizes it when it is scattered across different places or programs, for example two schedules, or two address books. The data manager is a portal from which users and applications access desired data, especially when the user is remote from the current environment. The point is that the user does not need to remember or care where the data is located physically.

The other function of the data module is the transcoding that filters the data and transforms it to suit applications or devices. This could happen when a program imports some data that does not have a consistent format. File-format transfer can also happen when data moves to a resource-limited mobile device. If that device does not have an appropriate program or enough resources to open the data, then the personal-data management module will transfer the source data to an appropriate format or reduce the quality of the data, such as reducing the resolution of a picture.

The interface module concerns the human-computer interaction (HCI). In a smart space, devices are pervasive and heterogeneous. Users are not restricted to keyboards, mice, and screens. Input or output may happen at different places with different devices. The user-interface module controls these things. For example, it manages input from the user explicitly or implicitly (no explicit command from the user); it transforms input data to fit the program and output data to suit the user's current device.

We have summarized the recurring features of pervasive-computing systems. These systems make solid contributions in that they bring the Weiser's vision of pervasive and ubiquitous computing clearer and closer. They also show us the recurring themes in terms of system design and construction. However, these systems are not complete and have some issues, such as isolation from each other, no standardization and so on. Section 4.3 discusses them in detail.

4.3 Insufficiencies in previous systems

Previous systems spend much effort on exploring various aspects of pervasive computing, and they usually stand alone. Inter-system integration is not often considered. Hence existing systems are mostly isolated from each other. They are not built on standards. Their short lifespan indicates that further evolvement and adoption may be difficult. Many of these systems also have a high barrier to entry because of particular system requirements, documentation incompleteness, and limited number of users. We consider these missing points from first-generation systems. Sections 4.3.1 to 4.3.4 elaborate on them.

4.3.1 Isolation

Existing systems are usually designed to solve one or a few aspects of pervasive computing, for example Aura features user-intention detection and proactive computing, while iRoom focuses on collaboration. They are often tailored for specific environments, such as offices or homes. These systems may be comprehensive but none is complete. This means individual systems would have to cooperate to paint a complete picture of pervasive computing.

However, existing systems usually make different assumptions about parts of a system while developing solutions. For example, Aura assumes the environment is often disconnected, so it incorporates the Coda file system for such disconnected operations, while Gaia assumes file systems should be associated with context, hence it includes CFS. Solutions from one system may not necessarily fit in the assumptions of another, which prevents the two systems from being merged into one. For example, SOUPA is an ontology framework developed for pervasive and ubiquitous computing, while Gaia uses first-order logic to deal with context. Besides, different systems usually have different views for users, objects, context, and so on. The mechanisms and techniques used in these systems are diverse. The programming languages for developing new applications may also differ from each other.

The result is that each system works within its own territory, but isolated from each other. The differences in architectures, constraints, interfaces, message types, and many other aspects prevent systems from interacting with each other. Interoperability is highly desired in pervasive computing, but it has yet to be achieved.

4.3.2 Insufficient integration platforms

One aspect of pervasive computing is seamless integration of software and hardware components into a unified system, so that human users can use it easily and consistently. Previous work in this direction is usually insufficient for two reasons. One is that the integration tends to provide managerial components to users and applications while hiding the individual service or device from them, such as the environment manager in Aura. The advantage is that applications can use abstract descriptions for services and devices and let the system do the mapping. The disadvantage is that it becomes less convenient and more cumbersome when applications or users want to access specific devices or services, such as a user controlling nearby lamps from his laptop. The desired approach is to provide the high-level interfaces for managerial components and also expose the low-level interface for devices and services to users and applications.

The other reason for previous integration platforms being insufficient is that they often lack a consistent view of users, applications, services, and devices. There is no consistent naming scheme for these entities. In many cases, previous systems focus on building applications using services and devices. It is convenient for users to use devices and services, but less convenient for devices to initiate an interaction with users without using an application. A desired platform should identify users, devices, and services consistently, not only in local environments, but also across them. This platform should also allow all entities to interact with each other in an easy way.

4.3.3 No standards

Pervasive computing is a pioneer field compared with distributed and mobile computing. Researchers have explored many possibilities of it and made much progress. At the same time, none of the previous projects has prevailed in practice. One of the reasons is that they are not based on standards. Existing projects use their own formats for data representation, communication, accessing context, and so on. It is difficult for systems to interact.

It is important to introduce standards into pervasive-computing systems. It enables interoperability among them. It also ensures compatibility when systems evolve. With standardization, different systems may be combined into one that is truly ubiquitous and pervasive.

4.3.4 High barrier to entry

A pervasive-computing system usually involves many components and groups of users. In order to make such a complex system pervasive, the barrier to entry has to be low for both professionals and non-professionals. Early systems are often built in order to explore novel ideas. They have core components but may lack finishing touches. In order to turn them into realistic systems, developers may have to deploy or develop more facilities or components. For example, in Cooltown, a beacon has to be in place for every object of interest.

Existing systems tend to be targeted to professionals, such as researchers and developers, hence the system may be too complex for non-professionals to perform simple tasks. For example, iRoom addresses data moving, control moving, and application collaboration, while a home user may just want to interact with surrounding devices most of the time, which may not necessarily utilize the core functions provided by iRoom. We believe a system should be designed to facilitate the implementation of new ideas for professionals, as well as to make basic tasks easy for non-professionals.

Besides, existing systems are used mainly by a small number of researchers, which means limited community support. Documentation is often insufficient. Few programming languages are supported, such as C++ and Java in Gaia. All these create a high barrier to entry, which prevents developers from prototyping quickly and extending systems with new functionality. These barriers also hinder non-professional users from using the system and adopting it in their daily lives.

Isolation, insufficient integration platforms, the lack of standards, and the high barrier to entry are major common deficiencies in previous research. They are also related; for example, standardization helps reduce the barrier to entry and promote interoperability.

With regard to this thesis, the survey on previous systems provides us insight into other pervasive-computing systems. The recurring themes and insufficiencies guide the design and construction of our non-intrusive-computing system. The next chapter describes a non-intrusive-computing system.

Chapter 5

A non-intrusive-computing system

The goal of non-intrusive computing is to help users manage interactions in a pervasive-computing environment, so that non-deliverable interactions do not disturb users, while deliverable ones are delivered to users appropriately. Given the model and the discussion in Chapter 2, we need to address a number of problems to make non-intrusive computing possible. For example, context needs to be collected and exposed to the model, there should also be a security mechanism to protect users from malicious interactions, and so on. The non-intrusive-computing system not only implements the model, but also solves these system issues.

In this chapter, we describe such a system. Inspired by Jabber, an open-source IM system, which will be detailed in Section 5.2, our non-intrusive computing system consists of middleware and application logic for filtering and delivering interactions. These two major functions are implemented based on system abstractions, services, and facilities, such as a general communication mechanism among different entities, a consistent naming scheme, and so on. In addition, the system is able to span multiple administrative domains and federate with other similar systems.

This chapter has three sections. Section 5.1 is an overview of the system. Although the system is inspired by and built on Jabber, the discussion is kept on a high level without using Jabber terms. The focus is to explain the system structure, its functional components, and their compositions. Section 5.2 describes the enabling technology, Jabber, and how the system components and requirements map to Jabber features and protocols. Section 5.3 discusses details of our implementation, how various pieces fit into the overall system, and the system overhead. Section 5.4 studies the Code Blue scenario to show

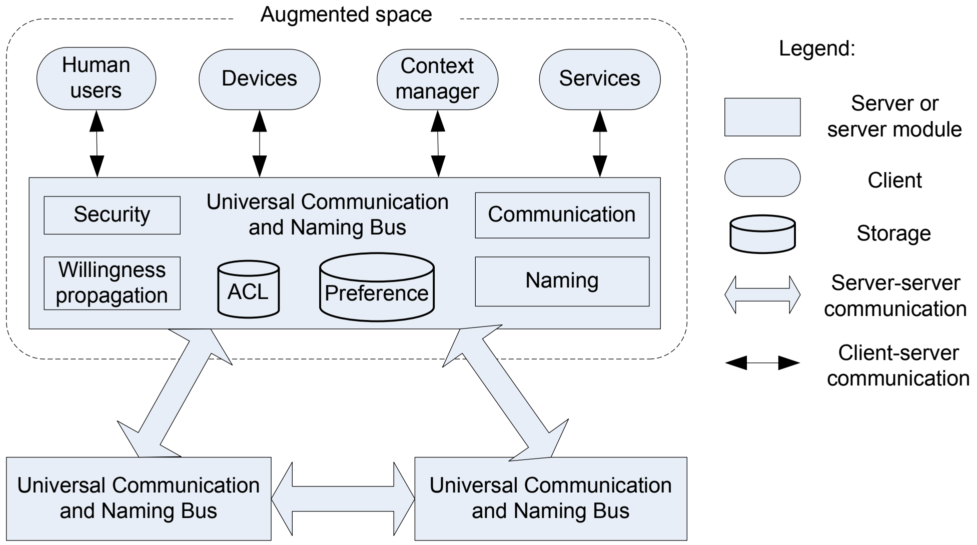

Augmented space
Human users
Devices
Context manager
Services
Security
Universal Communication and Naming Bus
Communication
Willingness propagation
ACL
Preference
Naming
Universal Communication and Naming Bus
Universal Communication and Naming Bus
Legend:
Server or server module
Client
Storage
Server-server communication
Client-server communication

Figure 5.1: Overview of the non-intrusive-computing system

how non-intrusive computing may be applied.

5.1 Overview of the non-intrusive-computing system

In Section 1.2, we briefly mention the design principles of our system. In general, the system should be a platform that can integrate hardware and software components. The system should also have a naming scheme for entities. It should also provide context and security support. When the system is built, we should use standardized technologies and reuse existing components. Figure 5.1 is an overview of our system. It follows these principles.

The simplest non-intrusive computing system starts from a single system with a client-server architecture, as shown in Figure 5.1. Then multiple such systems federate together through server-to-server communication and form a larger distributed system. A single system usually integrates with a smart space and manages local clients, such as users, devices, and services.

In a simple system, there are a server and a number of clients. The server provides core facilities and functionalities, such as security and naming, and acts as a communication bus for clients. Clients from different systems can also communicate with each other with the server-server communication. Clients may perform various functions in the system. They can be pure messaging clients, such as Google Talk, device clients, such as a printer or a sensor, or service clients that provide services that the server does not have and are considered server extensions, such as the context manager.

The server is called UCNB (for Universal Communication and Naming Bus). As indicated by the name, the server provides naming and communication infrastructure for the system. The communication mechanisms are versatile, including message-oriented communication, streaming-voice communication, remote procedure call (RPC), ad-hoc commands, and so on. The naming scheme is similar to that of electronic-mail systems. Each UCNB server has a unique domain name; clients have their names concatenated to the server name. Each UCNB ensures that there are no naming conflicts within its own domain.

Other than naming and communication, the UCNB also has a security module. Clients have to be authenticated by the server to use the system at all. The server also maintains an access control list (ACL) for each client, as shown in Figure 5.1. Basically, an ACL contains a list of clients and specifies their rights to communicate with the ACL owner. The ACL may be divided into a number of groups, with group-based authorization. Clients can also interpret their ACLs in different ways. For example, one client may grant communication for all clients on its list, while the other may deny communication for those on the list. More importantly, ACLs are the basis for propagating willingness, which is also the presence model in Jabber (Section 5.2).

As discussed in Section 2.3, the filter stage of non-intrusive computing compares the unwillingness and the importance. The sender specifies the importance and the receiver specifies the willingness. Thus, the willingness needs to be propagated to the sender so he can perform the comparison. If the willingness changes, the new value should also be propagated. For a given user, the UCNB server only broadcasts his willingness and updates to other users on his ACL.

We choose the sender to decide deliverability of interactions and filter them, as opposed to the receiver, the UCNB server, or a third party, such as a service client. This is based on the assumption that we are dealing with time-coupled interactions, and consideration that the server should be simple and generic. First, suppose the receiver client is to do the filtering. All interactions will be transferred from the sender side to the receiver side. If an interaction is indeed intrusive, the receiver client either holds or deletes it. However, since it is time coupled, it is of little interest to the receiver if it cannot be delivered immediately. So the system makes unnecessary transmissions. Second, we do not let the server make the filtering decision, because filtering is a specific high-level function that should not be the server's responsibility. Third, although a third party may be capable of filtering interactions, it requires the importance and willingness to be propagated, then the decision propagated back. Obviously, this approach is cumbersome. So having the sender decide matches intuition and may achieve some load balancing, as the filter and delivery stages are implemented on the sender and the receiver respectively.

There is storage for various preferences in UCNB. It is a generic data-storage mechanism, and clients usually use it to specify such preferences as how to handle the delivery of an interaction. For example, a user may prefer to deliver an interaction to her laptop instead of her cell phone if both satisfy all other criteria in the delivery stage. As it is generic storage, clients can use it store any specific data.

Notice that the server does not provide context service itself, leaving that responsibility to a separate client. This arrangement makes the server robust and simple. The system is also extensible, given the split between the server and the clients. We can easily replace or upgrade the current context manager with a new client without affecting the whole system. The client-server and server-server communication interfaces are standardized, which increases the system extensibility further.

Next, we describe Jabber that inspires and enables our non-intrusive computing system.

5.2 Jabber

Jabber is an extensible IM system. More precisely, Jabber is a set of streaming XML protocols and technologies that enable any two entities on the Internet to exchange messages, presence, and other structured information in close to real time.

5.2.1 Jabber overview

Jabber is an open-source instant-messaging-and-presence system. The Internet Engineering Task Force (IETF) has standardized the core Jabber protocol as the eXtensible Messaging and Presence Protocol (XMPP) [45]. Jabber extends traditional IM systems in the following ways.

- Traditional IM enables message transfer between two people. Jabber enables this between any two entities. An entity can be a person, a device, a software service,

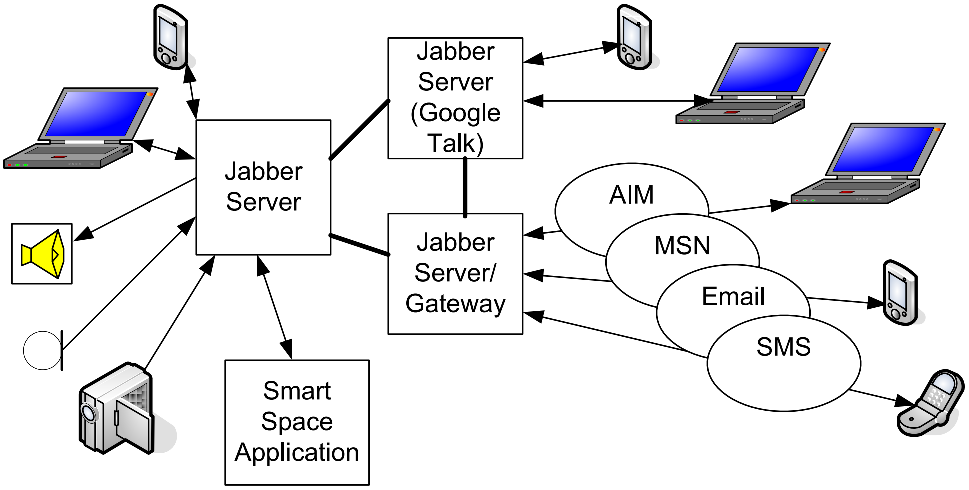


Figure 5.2: Jabber system illustration

or a conference room. Each entity has a unique Jabber ID (JID). Most of the JIDs take the form `user@domain`. For more information about JIDs, see the XMPP Core specification [45].

- The Jabber system has gateways to other environments, such as other IM systems, electronic mail, and short message service (SMS), so that a Jabber user can talk with an MSN messenger user, electronic mails can be forwarded to a Jabber user, a Jabber user can send a short message to a cell phone, and so forth.
- Jabber defines a set of enhancement protocols on top of XMPP, such as ad-hoc commands, Jabber RPC, pubsub, and so on.
- The architecture of the Jabber system is distributed. A Jabber server has a number of registered clients. Clients on the same server interact through that server; clients on different servers interact through server-to-server communication.

Figure 5.2 illustrates the concept of the Jabber system. In the middle of the figure, there are three Jabber servers and gateways. Each server has a number of Jabber clients. These clients can be regular desktop/laptop users or PDA users. The clients can also be sensing and actuating devices, such as a web camera or a speaker. In addition, the clients can be software applications, which may provide some service to the system, such as the context-manager application in the figure. We can also see that Jabber can connect with other systems directly or thru gateways. For example, it is able to federate with Google talk directly, and communicate with electronic mail systems, SMS systems, and traditional IM systems, such as AIM or MSN messenger, through the corresponding gateways.

Next, from Section 5.2.2 to 5.2.5, we discuss some of the Jabber features relating to the non-intrusive-computing system.

5.2.2 Jabber naming

Jabber has a naming scheme for both users and objects, which are all Jabber entities. Jabber uses Jabber IDs (JID) to identify entities. A JID is similar to an electronic-mail address. For example, a JID for Hao Chen is `h8chen@jabber.uwaterloo.ca`, and a JID for a printer in the Shoshin lab is `hplaserjet@jabber.uwaterloo.ca`. If a user has multiple devices, such as a laptop and a cell phone, we can identify them as resources of the user. So the JIDs for Hao's laptop and cell phone are `h8chen@jabber.uwaterloo.ca/ThinkPad` and `h8chen@jabber.uwaterloo.ca/motorola` respectively. In principle, any entity is allowed to have multiple resources. For example, a printer has printing options for color and black-white. We can use, for example, `hplaserjet@jabber.uwaterloo.ca/color` and `hplaserjet@jabber.uwaterloo.ca/blackwhite` to represent them.

5.2.3 Jabber communication mechanism

Although Jabber is an instant-messaging-and-presence system, it enriches the communication support beyond chat to many other interaction semantics, such as Jabber RPC [XEP-0009][1], ad-hoc commands [XEP-0050], streaming audio and video [XEP-0166], and so on. Interactions can happen between any two Jabber entities and in any format supported by Jabber.

In Jabber remote procedure calls (RPC), the remote server is a Jabber entity, which we call an RPC server to differentiate it from other Jabber servers. It has procedures that can be called by remote clients, also Jabber entities. A client sends an XML message that encodes the server's JID, method name, and parameters to the RPC server. The RPC server processes the request and returns an XML message to the client. In fact, Jabber RPC implements XML-RPC [65] over XMPP.

[1] The bibliography for this protocol is [66]. For other XEPs, please refer to the same entry.

A Jabber entity can implement ad-hoc commands; other Jabber entities can invoke these commands with XML messages. For example, a smart light might support such ad-hoc commands as “turn on,” “turn off,” “dim,” and so on. Ad-hoc commands are similar to Jabber RPC in that they support interactions between two Jabber entities. But ad-hoc commands fit better than Jabber RPC in the situation where human users are involved and ad-hoc commands are not known beforehand. In other words, the client of a human user can send an XML message and discover what commands are supported, and then present an interface for the user to pick one to execute.

Before executing the ad-hoc commands, the two entities need to exchange arbitrary command data. Jabber defines a protocol, Data Forms [XEP-0004], to describe generic data. It is useful not only for ad-hoc commands, but also for any situation that exchanges arbitrary data. It assumes a style of forms-based request and response between the entities.

Most forms of Jabber communication have one or more Jabber servers involved, depending on whether two communicating entities are on the same server. This is a type of client-server communication. Jabber also supports peer-to-peer communication, which is mainly for streaming audio and video sessions. When two entities want to start such a session, they first negotiate, through Jabber servers, data coding/decoding (CODEC) protocols and the transport candidates, such as IP addresses, ports, transport protocols, and so on. Then the two entities transfer multimedia data directly using the agreed CODEC and the transport candidate.

In addition, Jabber has a pubsub facility [XEP-0060], in which both publishers and subscribers are Jabber entities. A publisher publishes a message item to a topic, and then all the topic subscribers will receive the newly published item. This communication mechanism disengages publishers and subscribers. A publisher does not need to know who will receive the message, and a subscriber does not need to know who sent it. This pubsub mechanism is ideal for context-aware systems, where context providers and consumers can be associated and disassociated dynamically. For dynamic context or context changes, the consumers do not need to poll individual providers or check with a context manager periodically. Instead, the pubsub system pushes new items to all subscribers instantly. Essentially, the pubsub facility provides a time-coupled and reference-decoupled communication channel that is common in pervasive-computing environments.

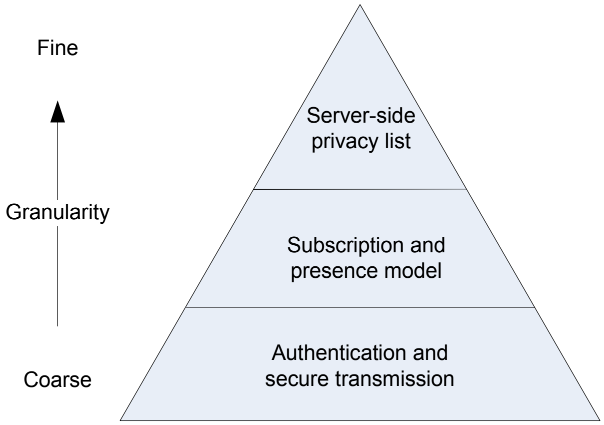

Fine
Granularity
Coarse
Server-side privacy list
Subscription and presence model
Authentication and secure transmission

Figure 5.3: Jabber security

5.2.4 Jabber presence and security models

Similar to other instant-messaging systems, Jabber has a presence model that allows entities to subscribe to presence and updates from each other. This provides an opportunity for an entity to authorize other entities at its own discretion, hence the presence model also becomes part of the security model. Jabber supports security in three layers (Figure 5.3).

At the bottom layer, it uses TLS [17] for secure data transfer between clients and servers, among servers, and among server components inside a server. It also uses SASL [36] to support authenticating users. These ensure the data stream remains intact and users are who they claim to be. An implementation of the Jabber server can support various authentication modules. For example, the Openfire Jabber server (`http://www.igniterealtime.org/projects/openfire/index.jsp`) supports MySQL, PAM (pluggable authentication modules), and others.

The next layer of Jabber security is achieved by the presence model. It works as follows. Entities may subscribe to each other's presence. If an entity updates its presence, the Jabber server will deliver the updated presence to all other subscribers. The presence model is used for the receiver to transfer his willingness to the sender in the filter stage (see Section 2.2). The presence model also helps protect users' privacy. If one entity wants to know the presence and availability information of another entity, it has to subscribe and obtain consent from the other entity. If the subscription request is granted by the first entity, the Jabber server will disclose the presence information and any updates of the subscribee to the subscriber. In Jabber, the presence information may be extended to include user activities [XEP-0108], mood [XEP-0107], locations [XEP-0080], tunes [XEP-0118], and so on, to describe the user's status in detail. The Jabber subscription relationship could be one-way, two-way, or none. Every entity has a roster that lists JIDs with which it has subscription relationships. Users can define groups inside the roster to categorize the contacts.

The third layer is the server-side privacy list [XEP-0016] that provides a fine-granularity security mechanism. Server-side privacy lists allow users to deny or allow certain XML messages based on JIDs, roster groups, or subscription status. There are three kinds of XML messages. The first kind is the Message stanza, with which users chat with each other or send messages. The second is the Presence stanza with which entities exchange presence information. The third is the IQ (Info/Query) stanza, which is for information, query or configuration purposes. The server-side privacy lists define rules or policies for the server to process XML stanzas. In detail, the Message, the Presence, and the IQ stanzas may be allowed or blocked based on specific JIDs, a user's roster groups, or a user's subscription types.

With privacy lists, a user can specify list rules, for example, denying all of the Message stanzas from a list of JIDs, or blocking the inbound/outbound Presence from/to a particular group, or all of the IQ stanzas from those entities with which the current user has a "From" subscription relationship. "From" means that entity is subscribed to the current user, but not the other way. Jabber defines a few subscription relationships, such as "To," "From," "Both," "From + Pending out," and so on. For a detailed explanation of these, refer to the Jabber protocol, XMPP IM [44]. Note that this server-side-privacy-list is unable to allow or block stanzas in further granularity. For example, it cannot control communication based on the namespace of an IQ stanza.

When a server is going to send an XML stanza to the user, the server first checks the privacy lists to filter undesired messages for the user. The privacy lists are flexible. Users can specify multiple lists and they can modify the lists at any time.

5.2.5 Jabber storage

Jabber allows its clients store arbitrary XML stanzas on the server. The data can then be retrieved by the same client [XEP-0049]. Aside from this general-purpose storage facility, Jabber can also store shortcuts to various services and resources for users [XEP-

0048]. These shortcuts may be in two forms, a URI or a Jabber conference room. These server-side storage facilities allow clients to save their preferences when needed.

5.2.6 Jabber and the non-intrusive-computing system

We mention that our non-intrusive computing system is inspired by and based on Jabber technologies. We choose Jabber because its design, architecture, and features match the requirements of non-intrusive computing.

The goal of non-intrusive computing is to control interactions in a ubiquitous-computing environment. Interactions are generic and not in a particular format. So first of all, the system has to support different communication mechanisms. Jabber, as an extended IM system, provides a rich selection of communication methods (see Section 5.2.3).

Second, non-intrusive computing manages interactions regardless of the senders and receivers. That is to say they do not necessarily have to be human users, especially for the senders. In pervasive computing, many smart devices or programs are able to initiate interactions. This requires the system to support users, devices, and software components universally. Participants in Jabber systems are not confined to particular users or devices. Instead, any entity that implements the XMPP-Core and XMPP-IM protocols can establish a connection with a Jabber server and interact with other entities on any Jabber server. The open architecture and standardization are strengths of the Jabber platform, and ease its adoption.

Third, one of the core ideas of non-intrusive computing is filtering non-deliverable interactions by comparing the importance with the importance threshold (unwillingness). This requires these two factors to be available at the same time and in the same place. The IM presence model supports this well by propagating presence updates to all subscribers. Jabber even has an enhanced presence model that carries more information, such as location [XEP-0080], activities [XEP-0108], mood [XEP-0107], and so on. Although as described in the protocol [XEP-0163], these pieces should be delivered to subscribers through the pubsub facility instead of the native presence-model implementation in the Jabber server, it makes no difference to non-intrusive computing in this regard. Note that all this extended presence information does not provide values for willingness directly. Instead, they offer context that may be used by a system to infer willingness more accurately.

Jabber's various facilities, such as private XML storage (Section 5.2.5) and security support, are suitable for non-intrusive computing as well, as we can see that in Section 5.1, there is the ACL and Presence storage in the universal communication and naming bus.

Other than these compatibilities, Jabber has a number of other advantages that make it viable for non-intrusive computing.

- Jabber has an increasing user base, and has become more popular with the introduction of Google talk. Jabber also has prompt and quality community support.
- Jabber has a rich set of resources including servers, clients, and coding libraries, which provide a low barrier to entry for both developers and users.
- Jabber uses XML to communicate among entities, which enables us to leverage existing XML technologies, such as XQuery [11].

Because of all these merits, it is reasonable and practical to choose Jabber and build a non-intrusive-computing system on it; the detail is in the next section.

5.3 System implementation

We reviewed the architecture of the non-intrusive system in Section 5.1 and explained the Jabber framework in Section 5.2. As we know, Jabber provides many mechanisms, protocols, and software, some of which may be used directly in our system, such as the Jabber server, its security mechanisms, the communication protocols, the private XML storage, and so on. However, to construct a full system for non-intrusive computing, we need to develop some new components. There are two major parts missing. One is the context manager; the other is incorporating the comparison model in the filter stage and the selection criteria in the delivery stage into the system. Sections 5.3.1 and 5.3.2 describe these two parts respectively.

5.3.1 Ontology and context manager

In the modeling of non-intrusive computing, we use context in both the filter stage (Section 2.3) and the delivery stage (Section 2.4) to refine the parameters and achieve better results. At that time, we simply used context as a system support and assumed its existence. In order to realize that assumption, two issues about context should be considered. The first is the format for representing context information. The second is how to manage context so that it can be accessed easily by users and applications.

We choose the Web Ontology Language (OWL) [50] to represent context data. An ontology describes concepts and their relationships. OWL was developed by the World Wide Web Consortium (W3C) to describe and process information on the web. The underlying data model is the Resource Description Framework (RDF) [51]. RDF describes knowledge using triples of a subject, a predicate, and an object. The object in one triple may be the subject of another. Hence, RDF data can form a graph. Using web ontologies is a significant trend, as both OWL and RDF are open and standardized by W3C. Furthermore, they are designed to be exploited by computers. Once data is in the form of ontology instances, any entity that understands that ontology is able to process the data. OWL is written in XML. Proprietary formats tend to isolate a system, hence reducing its usefulness.

In order to access ontologies, Jena [26] is used. Jena is an open-source programming framework for web ontology applications. Jena has APIs to read and write OWL data. More importantly, Jena has a built-in SPARQL [52] inference engine.

SPARQL is a language for querying RDF data, defined and standardized by W3C. The name is a recursive acronym for "SPARQL Protocol and RDF Query Language." Since OWL uses RDF as its data model, and RDF instances are expressed in the form of triples, SPARQL queries are based on the triple pattern. At the same time, SPARQL queries are similar to SQL queries for databases. For example,

```
select ?employee
from employee.owl
where
{
  ?employee age ?x .
  ?x greater_than 40
}
```

This query returns employees who are older than 40. ?*Employee* is the variable containing the return values. ?*X* is a temporary variable. *Age* and *greater_than* are predicates.

Each clause in the *where* portion of the query follows the (subject, predicate, object) pattern. SPARQL queries may ask for any elements in the triple, i.e., a subject, a predicate, or an object, or may query multiple elements at the same time. The following query returns all triples in the RDF graph for employees.

```
select ?subject ?predicate ?object
from employee.owl
```

A SPARQL query can also use multiple temporary variables, but none of them is returned in the results. The result of a SPARQL query consists of values for each selected variable. If there are multiple results that satisfy the *where* condition clause, these results are all returned. SPARQL results may be transferred over the network, in the XML format defined by W3C [53]. More information about SPARQL and its XML format can be found at their websites.

We further adopt the SOUPA ontology (Standard Ontology for Ubiquitous and Pervasive Applications [15] as the basis for context in our system. SOUPA is designed to model and support pervasive-computing environments and applications. It includes general descriptions and relationships about basic elements in a pervasive-computing environment, such as persons, spaces, devices, and so on. Figure 5.4 (taken from [15]) overviews SOUPA. Each box represents an OWL class, which may include sub-classes. For example, *Device* has a subclass for cell phones. Classes are related through object properties. For example, the device class defines an object property, *hasUser*, whose subject is a device and object a person. SOUPA is currently the most comprehensive ontology for pervasive and ubiquitous computing. The complete SOUPA description is found on its website (`pervasive.semanticweb.org/soupa-2004-06.html`). However, despite the maturity of SOUPA, it is still necessary to extend it to support our system properly.

The space ontology in SOUPA includes two sub classes, *GeographicalSpace* and *GeographicalRegion*. Their objects are like DC3552D and the city of Waterloo respectively. The SOUPA space also defines some properties, such as *spatiallySubsumes*, *hasCoordinates*, and *controlledBy*. Conversion among different measurements and representations of spaces can be done. Clearly, SOUPA tries to provide general and complete vocabulary and concepts and to differentiate them. The device and person ontologies are similar.

Note that in a pervasive-computing scenario, it is important to relate things to form

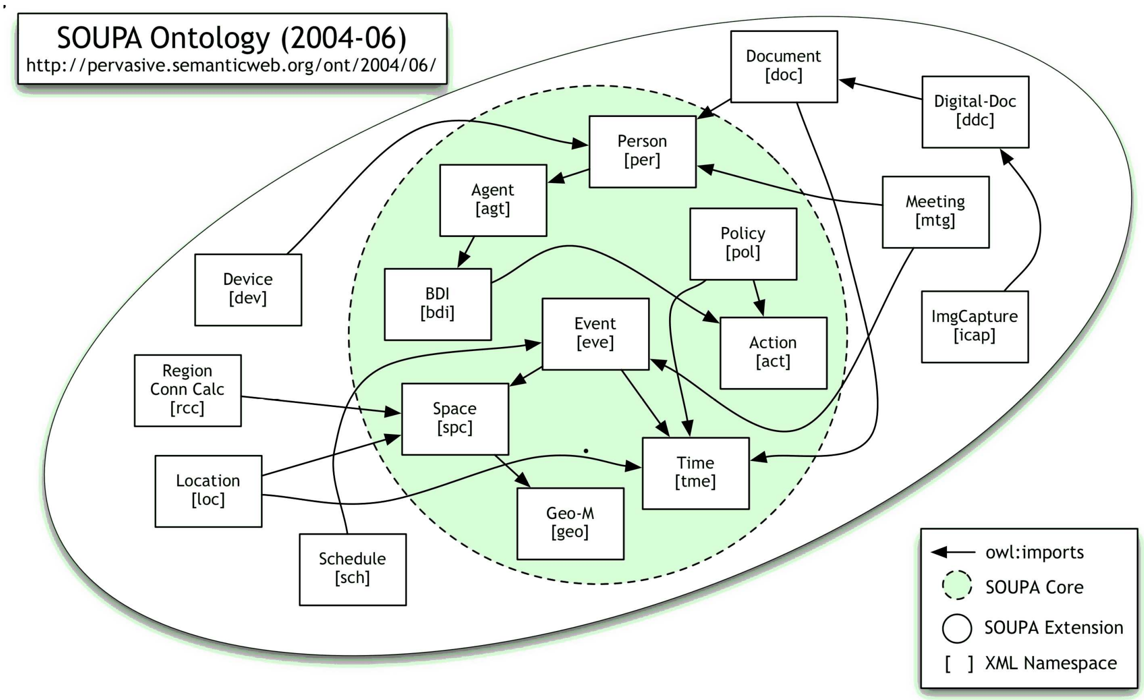


Figure 5.4: SOUPA ontology

context, such as combining users and locations to conclude who is where. SOUPA, however, is insufficient in this regard. Although, it does have the notion of a device's user, i.e., *hasUser*, it does not define where devices and persons are located. Both relationships are necessary in the non-intrusive-computing system, because selecting delivery candidates requires this knowledge to determine which devices are co-located with the receiver and if there are other users around. To address this problem, we add two properties. One is *spatiallySubsumesDevice*, whose subject and object are a *GeographicalSpace* object and one or more *Device* objects respectively. The other is *Location*, whose subject and object are a *Person* object and a *GeographicalSpace* object respectively. We also extend *Device* to a few sub-classes, such as *Printer*, *PC*, and so on. Necessary properties are also defined, such as notification methods and the effectiveness and overtness values.

We decide to choose SOUPA ontology to represent context in our system, and use SPARQL to query and access it. We now build a context manager that use both mechanisms to provide context to applications.

New Jabber entities can be implemented either as clients or as external server components. Clients use the protocols defined in "XMPP Core" [45] to connect to the Jabber server; external components use the "Jabber Component Protocol" (JCP) [XEP-0114] for the connection. These two types of entities are functionally similar. For a given service, we can implement it as either a client or a component.

However, there are some differences between the two. For clients, the Jabber server stores their contact lists and subscription relationships to other entities. If clients update their presences, the Jabber server will broadcast these updates to other entities that subscribe to these presences. On the other hand, an external component has to manage subscription and contact lists by itself. When it updates its presence, it has to send the updated presence to any subscribers. The Jabber server only forwards incoming and outgoing messages for external components. As to the authentication, clients each have different credentials, while components always use the same one. The naming convention for components is also different. Suppose there is a dictionary service connecting to the Jabber server `jabber.uwaterloo.ca`. Its JID might be `dict@jabber.uwaterloo.ca`, if it is a client, and `dict.jabber.uwaterloo.ca`, if an external component. An external component can act as a container to hold more entities. For example, a dictionary component contains two dictionaries, one for English and one for French. The JIDs for them are `english@dict.jabber.uwaterloo.ca` and `french@dict.jabber.uwaterloo.ca` respectively. Messages to both entities are forwarded to the same component, `dict.jabber.uwaterloo.ca`. In this case, the component has to manage communications for any entity it contains.

Hence, implementing an external component involves more work than doing so as a client, since the Jabber server does not manage rosters and subscription for components and contained entities as it does for clients. However, a component has the advantage of being able to host entities, which may be needed in some circumstances.

Our context manager may be implemented as either a client or an external component, but our choice is to implement it as an external Jabber component. The choice is more of a conceptual decision than a functional one, considering the context manager as an extension to server functions. Also, the component approach make it flexible to extend the context manager in the future, as we can add entities. For example, suppose the JID for the service manager is `icontext.jabber.uwaterloo.ca`, it may decide to have two separate entities, `query@icontext.jabber.uwaterloo.ca` and `update@icontext.jabber.uwaterloo.ca`, for retrieving and updating context.

Figure 5.5 shows the architectures of the context manager, iContext. There are two options. The difference is that the bottom option uses a pubsub server [18], while the top one does not. The pubsub server is also a Jabber component. In both architectures,

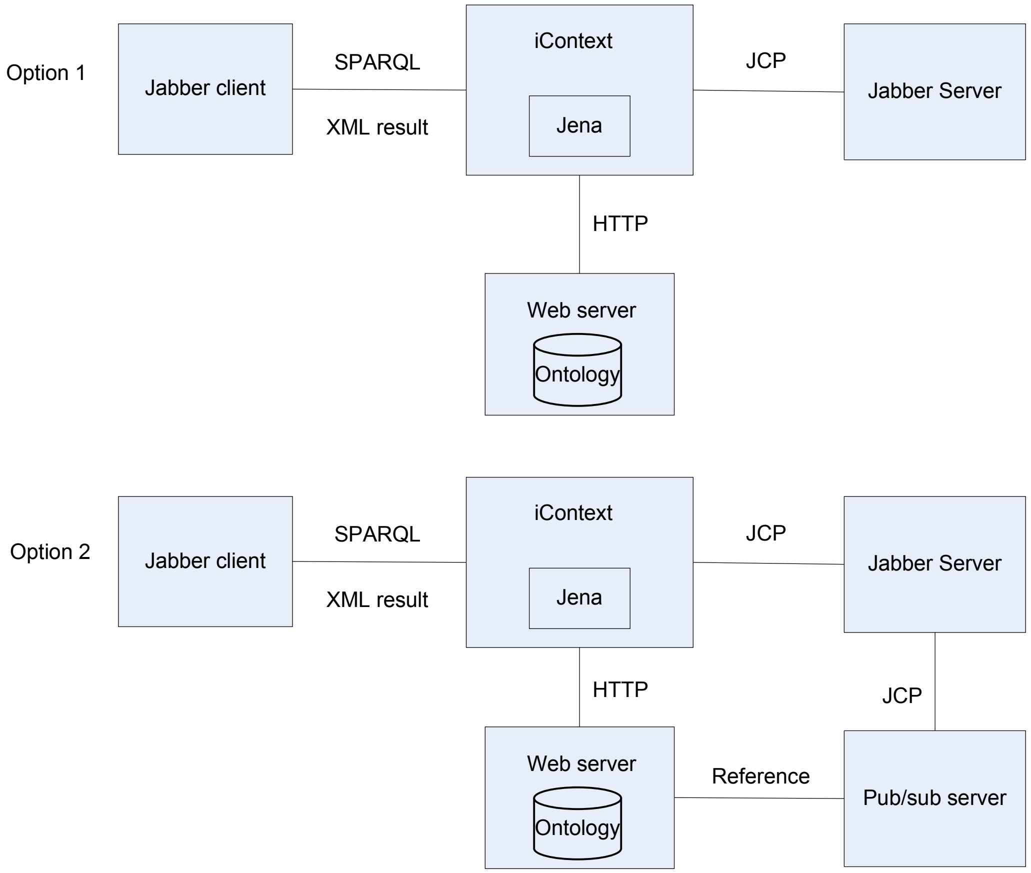


Figure 5.5: iContext architecture. Top: without using pubsub; bottom: with pubsub

iContext connects to a Jabber server using JCP. The ontology that describes the context is stored in a web server. In the bottom architecture, for dynamic context, the ontology only stores a reference, i.e., a node name, to the pubsub server. The actual context data is stored in pubsub under that node name. The advantage is that the pubsub server will notify the subscribers of any context changes.

In the top architecture, when iContext initializes, it reads context data from the HTTP server, then it waits for SPARQL requests from clients. After it queries the ontology, it returns results in the standardized XML format to clients. For example, in order to select location instances whose type is *GeographicalSpace* labeled *DC3552D,* we have:

```
type = <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
space = <http://pervasive.semanticweb.org/ont/2004/06/
        space#GeographicalSpace>
label = <http://www.w3.org/2000/01/rdf-schema#label>
SELECT ?x
WHERE
{
  ?x type space .
  ?x label "DC3552D"
}
```

The query returns:

```
<?xml version="1.0"?>
<sparql
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xs="http://www.w3.org/2001/XMLSchema#"
    xmlns="http://www.w3.org/2005/sparql-results#" >
  <head>
    <variable name="x"/>
  </head>
  <results ordered="false" distinct="false">
    <result>
      <binding name="x">
```

```
          <uri>
            http://grand.uwaterloo.ca/~h8chen/owl/
            mylocation.owl#DC3552D
          </uri>
        </binding>
      </result>
    </results>
  </sparql>
```

Note that to determine proper delivery candidates, a client may send multiple such query requests to iContext. Also, the requests and returns are encoded in XMPP traffic. In this case, they are the payload of Jabber "IQ" stanzas [44].

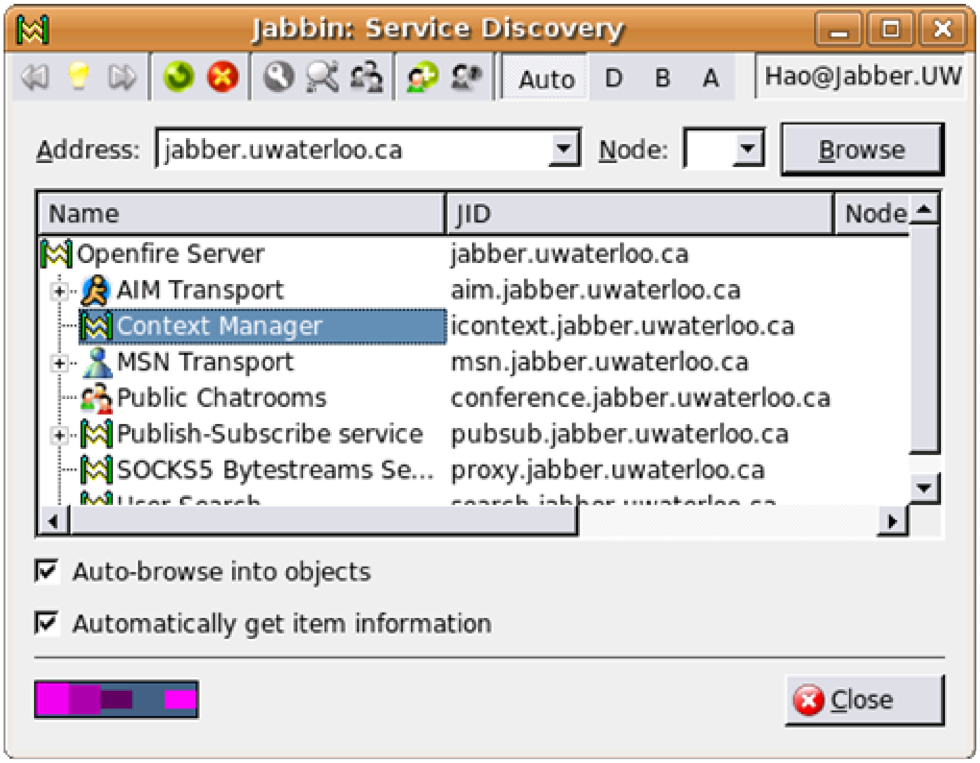
If the bottom architecture is used, the control flow is the same, except that the results returned by iContext may contain node names for dynamic context. Then the Jabber client needs to subscribe to this context on the pubsub server using the node names. With the pubsub mechanism, updated data will be pushed to the Jabber client without querying iContext. We implemented both architectures.

Figure 5.6 is a screen shot of service discovery in Jabbin (`http://www.jabbin.com/int/`), an open-source Jabber client. Each service is indeed an external component. We can see that the Jabber server is `jabber.uwaterloo.ca`. iContext has a JID of `icontext.jabber.uwaterloo.ca` and is currently connected to the server. Since the interfaces of iContext are all standardized, iContext can interact with other OWL ontologies, Jabber servers, and Jabber clients.

5.3.2 The models and Jabber clients

In the model of non-intrusive computing, the deliverability of an interaction is determined by comparing the importance with the unwillingness. Then there is a delivery model that selects appropriate devices and notification methods in the delivery stage.

As discussed in Section 5.1, we let the sender decide the deliverability of an interaction, and let the receiver select the proper delivery candidates, as shown in Figure 5.7. For this arrangement to work, the receiver sets his willingness. This willingness is propagated to the sender together with Jabber presence information. The sender is then able



Jabbin: Service Discovery

Auto D B A Hao@Jabber.UW

Address: jabber.uwaterloo.ca Node: Browse

Name	JID	Node
Openfire Server	jabber.uwaterloo.ca	
AIM Transport	aim.jabber.uwaterloo.ca	
Context Manager	icontext.jabber.uwaterloo.ca	
MSN Transport	msn.jabber.uwaterloo.ca	
Public Chatrooms	conference.jabber.uwaterloo.ca	
Publish-Subscribe service	pubsub.jabber.uwaterloo.ca	
SOCKS5 Bytestreams Se...	proxy.jabber.uwaterloo.ca	
User Search	search.jabber.uwaterloo.ca	

☑ Auto-browse into objects

☑ Automatically get item information

Close

Figure 5.6: iContext screen shot

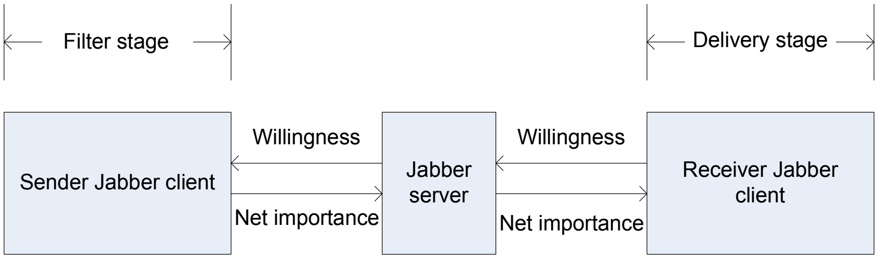


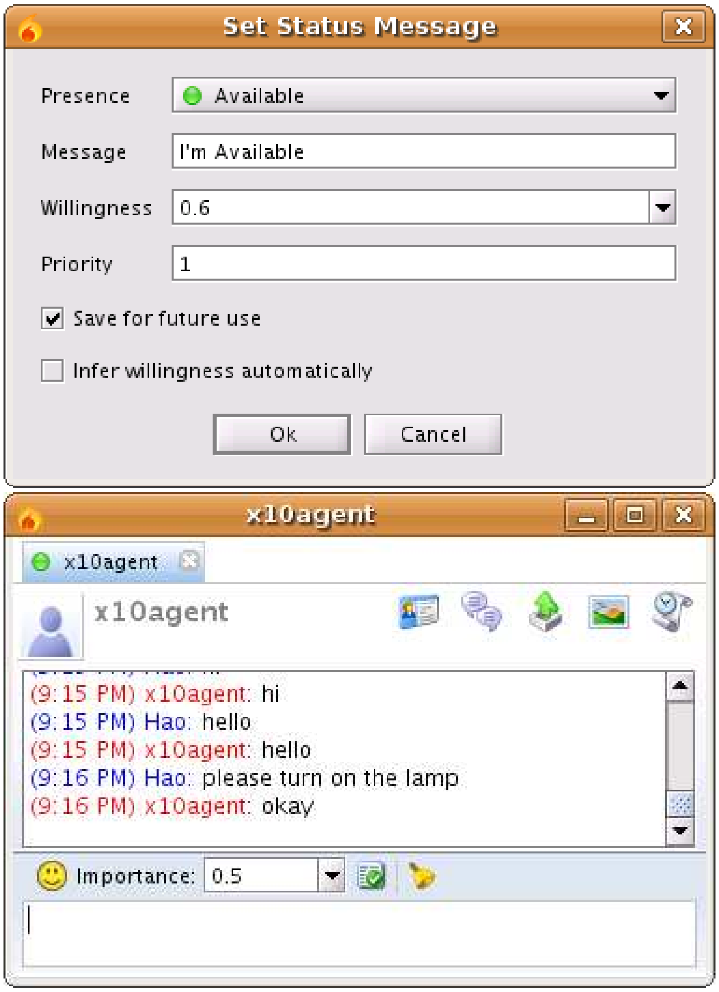
Figure 5.7: Model implementation

to compare the importance with the importance threshold. If the interaction is deliverable, i.e., the net importance is greater than zero, it takes place. At the same time, the net importance is forwarded to the receiver, so that he can apply the effectiveness criterion.

Note that, at the model level, the comparison function can be on the sender side, the receiver side, or in a third party. The only requirement is to have both the importance and the willingness together when the comparison takes place. At the implementation level, we may use any of the above three choices. The current implementation is inspired by the Jabber presence model of Section 5.2.4, which propagates the updated presence to all subscribing entities. This satisfies the requirement well. Also, there is an advantage in performing the comparison this way. If the interaction is non-deliverable, then the interaction body, such as a message, will not be transferred, saving unnecessary network traffic. If we were to make the comparison at a different place, such as the receiver's side or a third party, the system would send the interaction together with the importance value. If the interaction is determined to be non-deliverable, it will be dropped either at the receiver's side or the third party, which may waste network resources.

We modify Spark (`www.igniterealtime.org/projects/spark/index.jsp`), an open-source Jabber client, to implement the models instead of developing a new client, because reusing existing code for efficiency is one of the principles for our system. Spark has an open architecture and allows third-party plugins, from adding a button in the main window to reinterpreting incoming and outgoing messages. Our work includes a plugin and some source-code modifications. Figure 5.8 is screen shots of the modified Spark, where we allow senders and receivers to set the importance and willingness manually from combo boxes. Once the willingness is set, it is delivered to the presence subscribers as shown in Figure 5.9. Then at the sender side, we are able to filter interactions by comparing the importance and the importance threshold.

In a pervasive-computing environment where users' willingness tends to change frequently, this manual behavior may become a burden for users. Hence automatic setting is desired. To determine a user's willingness fully automatically and correctly is complicated and not the focus of this dissertation. However, as a proof of concept, we use a user's Jabber presence, the status of his office door, and entries of his calendar service, such as Google Calendar, to determine the willingness, since each of these has some willingness indication. We take the average of them for the final willingness. Jabber defines four levels of presence, "do not disturb (dnd)," "extended away (ea)," "away," and "chat." We map them evenly to $[0, 1]$, so that the willingness is 0.2, if the presence is *dnd*; and



Presence ● Available

Message I'm Available

Willingness 0.6

Priority 1

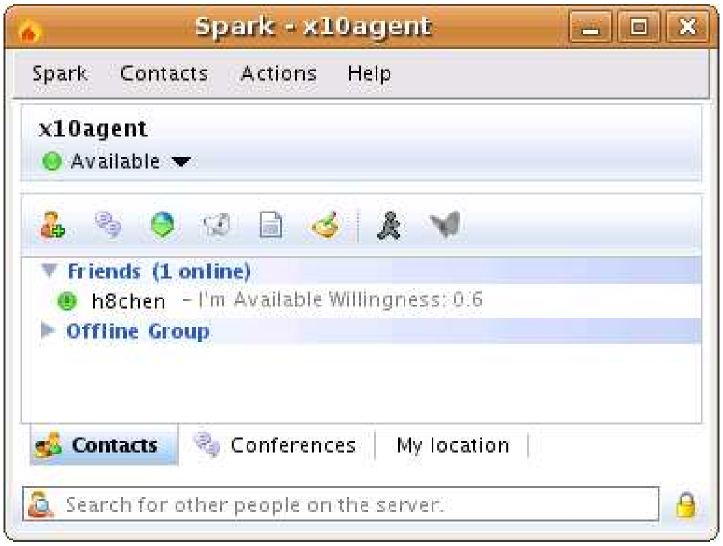
☑ Save for future use

□ Infer willingness automatically

(9:15 PM) x10agent: hi
(9:15 PM) Hao: hello
(9:15 PM) x10agent: hello
(9:16 PM) Hao: please turn on the lamp
(9:16 PM) x10agent: okay

Importance: 0.5

Figure 5.8: Spark modification. Top: receiver sets willingness; bottom: sender sets importance



h8chen - I'm Available Willingness: 0.6

Figure 5.9: Willingness propagated to the sender

0.4 for *ea*, and so on. The door status can be obtained from iContext, assuming there is a sensing mechanism, such as a contact sensor, which can monitor the door status, and update the corresponding context in the ontology. The door angle measurement can be normalized to range over $[0, 1]$. For simplicity, we only have 0 for willingness when the door is closed, and 1 when it is open. For Google Calendar, a user has to have a Gmail account, which can determine a unique URL feed where calendar data can be retrieved or updated. If there is an entry in the calendar, it sets the willingness to 0, otherwise, 1.

We prototype this inference mechanism in Spark (see the check box in the upper half of Figure 5.8). When the box is checked, it calculates willingness from the presence, the Google-calendar entry, and the door status. However, the door status is hardcoded as we did not deploy the necessary sensing equipment.

This method for setting the willingness is obviously coarse and may not be accurate. Even if there is context available and comprehensive inference mechanisms are used, the result may still be inaccurate. So we allow users to override the automatic willingness at any time.

Aside from modifying Spark, we also need to wrap some devices as Jabber entities. For example, if we decide to deliver a message to a printer, there should be a printer entity that receives the message and turns it into a printing command. There are many libraries for this type of functionality, such as Echomine Muse (`http://open.echomine.org`). A complete list can be found at `http://www.jabber.org/software/`

libraries.shtml. Typically, these libraries handle authentication and communication details and let developers focus on user interfaces and application logic.

5.3.3 System implementation

In the last two sections, we described iContext and the Spark modifications for the models, which are the major custom pieces of the non-intrusive-computing system. In this section, we explain how these parts are combined.

In Figure 5.10, three Jabber servers are inter-connected. Each of them connects to a few Jabber entities, such as the sender, the receiver, the printer, the cell phone, and so on. The context manager, iContext, connects to one of the Jabber servers as a Jabber external component. These are transport connections, which means they are actual routes for transferring data. On the other hand, the dashed lines indicate logical connections, which means the communication between two end points does not happen directly, but through physical ones.

When the system starts up, the context of each device is placed into the ontology either by the system administrator or by devices and their Jabber wrappers that can update it themselves, as is shown in Stage 0.

Both the sender and the receiver logon to their Jabber servers, which may or may not be the same one. The receiver reflects her willingness explicitly or the client software infers it automatically. The sender will obtain it through Jabber presence, assuming that the sender subscribes to the receiver's presence. Thus the sender can compare the importance of the interaction with the importance threshold when it attempts an interaction. If the sender decides that the interaction is deliverable, the interaction, such as a message, is delivered to the receiver client. The net importance is appended to the body element of the message's XML stanza, so it is delivered to the receiver as well. This process is Stage 1 in Figure 5.10.

In Stage 2, upon receiving a message, the receiver client first truncates the net importance from the XML message body and then sends SPARQL queries to the context manager, iContext. The queries ask for feasible delivery candidates at the receiver's location, assuming the receiver client knows the current location of the receiver. The iContext component executes the queries and returns XML results. The receiver client is then able to perform the effectiveness and overtness criteria using Algorithm 2.3. The last step is to forward the message without the net importance to selected delivery candidates.

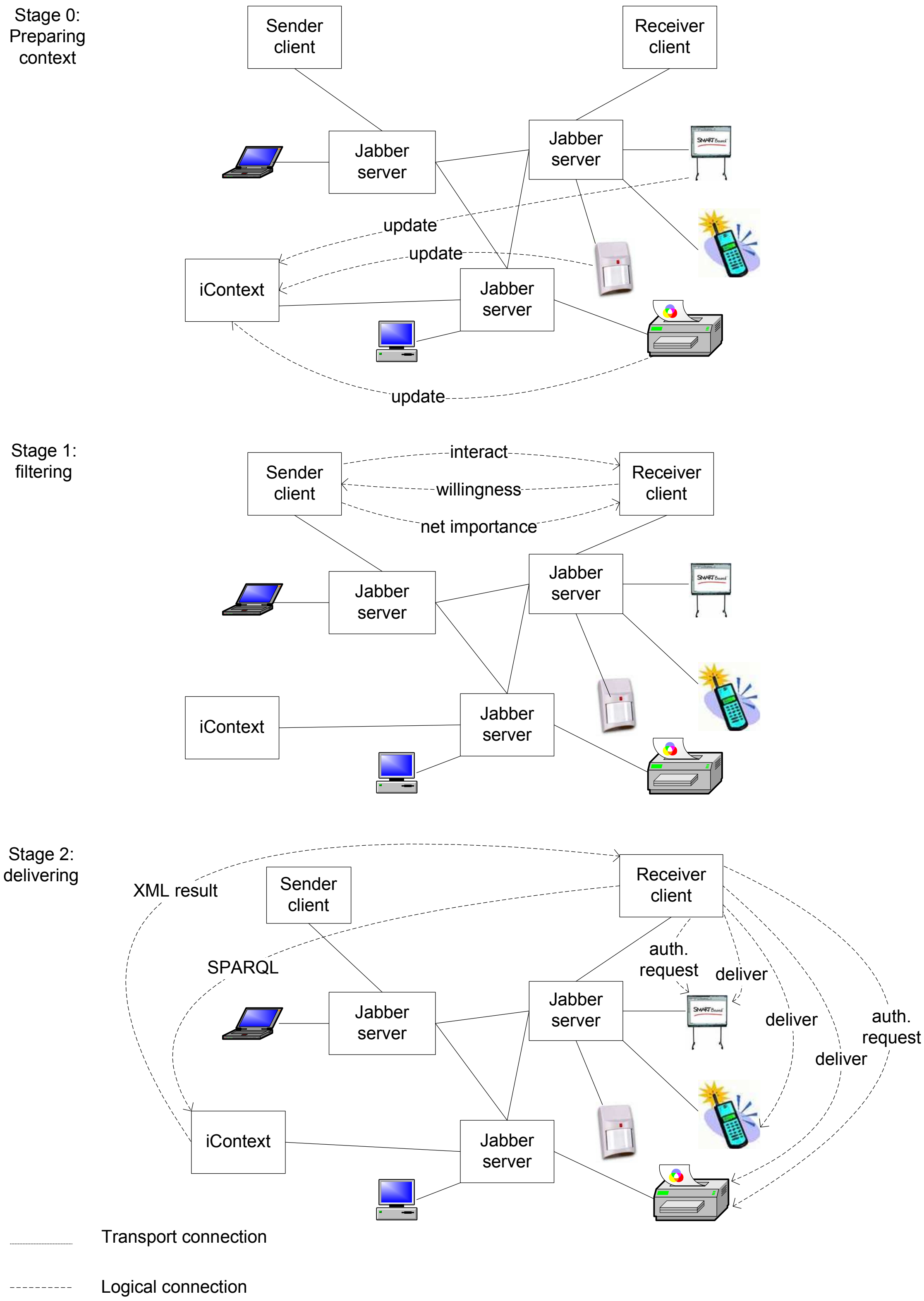


Figure 5.10: System summary

Some of the devices in this scenario are public, such as the whiteboard and the printer in Figure 5.10. To prevent users, such as newcomers, from abusing them, and also increase security, the receiver client has to be authenticated before it can deliver any interaction to those devices. The authentication makes use of the subscription model and the server-side privacy lists of these devices. A user first sends a subscription request to the Jabber wrapper of the device. If it is approved, the device's Jabber wrapper adds the subscriber to its privacy list, so that all further communication is allowed.

The above procedure may happen in one of two ways. If a user is known to an environment, such as Alice to her lab, then the system administrator may set up the subscription and the privacy lists statically, so that she is allowed to use all public devices at any time. If a user is new to an environment, for example, when Bob comes to Alice's lab for a meeting, then Bob has to be authenticated by the whiteboard in Alice's lab. There is a password for the whiteboard that Alice can tell Bob to include in his subscription request to the whiteboard, to establish the subscription.

Once the receiver client determines the final delivery candidates, and authenticates to public ones, it delivers the interaction.

If there are changes in context, such as a sensor sensing movement, a printer load becoming heavy, or no available display space on the whiteboard, information is updated and pushed by devices and their wrappers to context subscribers through the pubsub facility. Then for new interactions, the receiver client retrieves updated feasible candidates.

The non-intrusive-computing system is built on top of a number of technologies, such as Jabber, OWL, Jena, and so on. It leverages these enabling technologies to achieve the goal of controlling interactions in a way that is not intrusive to the receiver nor to other people in the same environment. The system has a clear architecture and is highly extensible. The next section discusses the system overhead.

5.3.4 Overhead evaluation

The two major components in our system are the Jabber server and iContext. Jabber provides near real-time communication. It competes with other well-known IM systems, such as MSN Messenger, ICQ, and so on. Many Jabber servers have been deployed world wide. The latest version of the Openfire server has been downloaded $1,462,777$ times (http://www.igniterealtime.org/projects/openfire/index.

jsp) by September 18, 2008. Also, public data from May 19, 2008 shows that Jabber is scalable: the public server at www.jabber.org had 10,054 connected users and 2,954 connections to other Jabber servers. Note that the new version of the website does not provide such data anymore.

In terms of delivering a message, the overhead comes from querying the ontology. We use Jena 2.5.4 (http://jena.sourceforge.net/) as the SPARQL query engine. Our ontology is derived from SOUPA [15]. We added five classes for describing devices, such as smart phones, printers, and so on. A number of properties are also added to both devices and locations. There are 20 instances in each device class, assuming that there will be no more than 100 delivery candidates in a smart space in the near future, and three instances in a geographical-location class. All data is in two XML files, one for devices, the other for locations.

We evaluate two SPARQL queries, on a Centrino Duo 1.73*GHz* laptop, with 2*G* memory, running Ubuntu 8.04. The first one selects all devices of different sub-classes of class Device, as shown below. This query only needs the data file for the devices. We run the query 10 times (Figure 5.11, left). The average time to finish this query is 71.3ms; the standard deviation is 3.35.

```
TYPE = <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
SUBCLASS = <http://www.w3.org/2000/01/rdf-schema#subClassOf>
DEVICE = <http://pervasive.semanticweb.org/ont/2004/06/
         device#Device>
SELECT ?x
WHERE
{
  ?y SUBCLASS DEVICE .
  ?x TYPE ?y
}
```

The second query (see below) selects the effectiveness and overtness values and the JIDs of those delivery candidates that are in room “DC3552D.” This query accesses data from both files. The same as the first query, we run this 10 times. The average running time is 74.5ms, with standard deviation 3.2 (Figure 5.11, right).

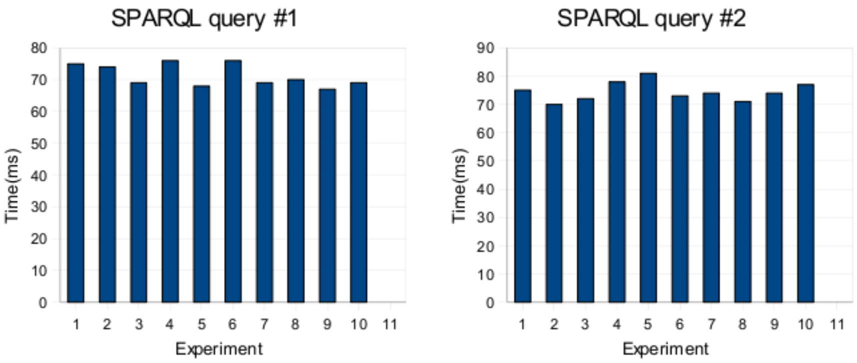


Figure 5.11: SPARQL query performance. Left: the first query, right: the second query

```
...
SELECT ?effect ?overt ?jid
WHERE
{
  ?loc TYPE geo_space .
  ?loc LABEL “DC3552D” .
  ?loc SUBSUME ?device .
  ?device EFFECTIVENESS ?effect .
  ?device OVERTNESS ?overt .
  ?device JID ?jid
}
```

As we can see that the running time in both cases is close and stable. Since we are dealing with human interactions, this overhead will not cause noticeable delay to people. Note that all data is preloaded, which could take some time, depending on the size and the network load.

So far, we have discussed the system design, enabling technologies, the implementation, and the performance evaluation. In the next section, we keep discussing how a non-intrusive-computing system could be applied to a different domain, and what applications may be built on top of it. We case study the Code Blue scenario.

5.4 Code Blue case study

Hospitals, especially large and busy ones, have emergency coding systems for responding to various urgent situations. When an emergency happens, such as a patient showing cardiac failure, the hospital broadcasts an announcement to call doctors, nurses, lab workers, and so on. The message usually includes some color code to identify the type of the emergency. Different color codes are often well established. For example, Code Black indicates mass casualty or public health threat; Code Blue is often dedicated for cardiac-related situations; Code White corresponds to pediatric medical emergencies, and so on.

With such a color-code system, a team may be summoned within a short period of time, such as minutes or even seconds. Typically, a public address system is used to deliver the message and capture people's attention. When it is being used, everybody in every floor of the hospital hears it. This approach is simple and effective. The drawback, however, is also obvious. It does not differentiate between people who should be involved and those who should not. This may cause disturbance since everybody is forced to listen to it. Imagine that a pediatric doctor is performing an operation on a child injured in an accident, and an announcement asks for cardiac specialists. The sound, to the pediatric doctor, is much like noise and may potentially distract his attention. It would be ideal if the emergency system delivers calls only to related personnel. Furthermore, existing color-code emergency systems usually do not perform other tasks than calling people. It would be better if these systems could coordinate the people and entities involved. For example, when a doctor is heading to a patient's room, the system could deliver the available medical results, such as X-ray and medication information to the TV in that room, which will help the doctor once he arrives and save some time in treating the patient.

With modern computing technologies, it is possible to acquire necessary context, determine who should receive urgent calls, who should not be disturbed, and help coordinate among doctors, nurses, medical resources, and so on. Non-intrusive computing provides an abstract framework to deal with interactions, which matches this problem well. Given proper context and implementation, we can design an intelligent emergency system that not only notifies proper receivers in a proper way, but also assists medical doctors in handling emergencies. Without loss of generality, we choose Code Blue as s case study for an intelligent system. The rest of this section describes the design of such

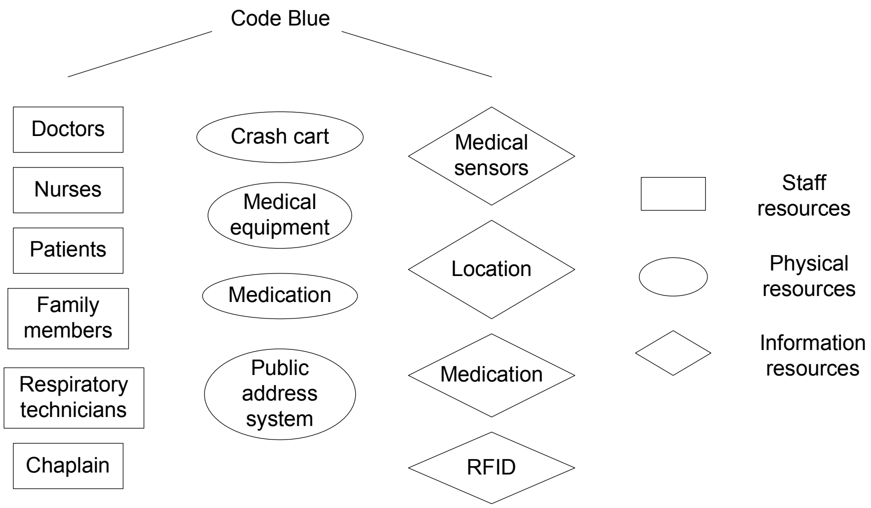


Figure 5.12: Code Blue

a system for Code Blue emergencies based on the non-intrusive-computing model.

5.4.1 Code Blue

A Code Blue system involves many components or resources. When an emergency is detected, the system tries to locate resources as completely and quickly as possible. For example, doctors and respiratory technicians should gather at some place; nurses should bring a crash cart; if the situation is potentially fatal, the system will also call a chaplain to be present, and so on. These resources are categorized as illustrated in Figure 5.12.

Information resources are critical to an intelligent emergency system. The system behavior depends on quality information resources. For example, medical sensors may be attached to patients, which can provide live data for the heart rate and the blood oxygen saturation. Once the sensor data shows that a patient is in danger, the system should start a Code Blue call. The location system will keep track of where the related doctors are. The system will call those who are less busy. In order to call them effectively and not too overtly, the system needs to know the available delivery candidates and the surrounding circumstances. RFID technology may be used to identify objects. Medical devices, such as crash carts, have RFID tags attached to them. In an emergency, the system may instruct nurses to bring these objects from the nearest place.

We can see that each of the information resources itself could be a complex system. For example, Lorincz *et al.* [35] use a wireless sensor network to propagate data. Loca-

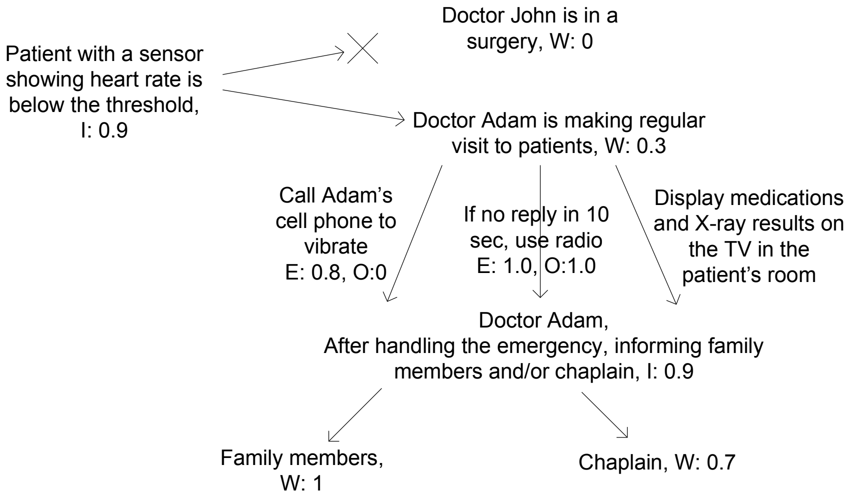

Patient with a sensor showing heart rate is below the threshold, I: 0.9
Doctor John is in a surgery, W: 0
Doctor Adam is making regular visit to patients, W: 0.3
Call Adam's cell phone to vibrate E: 0.8, O:0
If no reply in 10 sec, use radio E: 1.0, O:1.0
Display medications and X-ray results on the TV in the patient's room
Doctor Adam, After handling the emergency, informing family members and/or chaplain, I: 0.9
Family members, W: 1
Chaplain, W: 0.7

Figure 5.13: Intelligent notification

tion tracking may involve both indoor and outdoor technologies. The discussion in this section assumes the existence of these systems and focuses on combining non-intrusive computing with emergency handling.

Next we describes possible applications for this intelligent emergency system for Code Blue.

5.4.2 Application scenarios

In a Code Blue system, the most important thing is to call people to the emergency. This section first describes some intelligent notification applications, then an application for inpatients. Next, we discuss the possibility of engaging with other applications in a hospital. The goal is to show how to apply non-intrusive computing in such a system and how it may be used to improve handling of Code Blue scenarios.

The idea of intelligent notification is that when urgent situations happen, instead of using public address to notify everybody, the system decides who should be notified based on context.

Figure 5.13 describes such an application. Suppose a patient has some sensors attached to his body, which constantly provide data to the system. Once the sensor data falls out of the normal range, the system should intervene, either by calling for a nurse, a doctor, or a team depending on how severe the situation is. The sensor data determines the importance level of the interaction.

In Figure 5.13, the importance of calling for a doctor is determined to be 0.9. Suppose there are two doctors on duty at the moment. Doctor John is performing a surgery, so he is completely unavailable, with willingness 0. Doctor Adam is busy on regular visits, so the willingness is relatively low at 0.3. However, since the importance is high, the system decides to call Doctor Adam.

In terms of delivering this interaction, there are two options: one is through the cellphone, the other is using the public address system. The Code Blue system will first try to call the doctor's cellphone, because the system does not want to affect the patient who is with Doctor Adam. The cellphone is highly effective and not overt. If, for some reason, Doctor Adam does not answer the phone in a few seconds, the system will use the public-address system.

In the meantime, the system gathers all the information about this patient, such as X-ray, ultrasound, medications, and so on, and sends them to the patient's room. This information will be shown on the TV once the doctor arrives. In this case, the doctor does not have to go back to the nursing station to pick up the medical record. This will save time and effort for the doctor.

Afterwards, the system may inform the family members of this patient who are waiting outside about any progress. This gives peace of mind to the family members. If unfortunately, the patient is about to die, the system calls for the chaplain, who will come to perform the necessary ceremony.

In summary, this application shows two things. First, if possible, the system will try to avoid unnecessary disturbance by using cellphones. Second, the system makes the patient information available to the doctor in an automatic fashion, which not only helps the doctor diagnose the patient, but also saves some time which may be crucial in emergent cases.

Note that this example assumes a number of values for importance, willingness, effectiveness and overtness. The user study in Section 2.5 suggests that there are some discrepancies between our assumed values and average values from the survey questionnaire. Hence, these values are intended to demonstrate the application scenario, and do not necessarily mean they should be set this way, which is also the case for other applications later in this section.

Another application we describe there is how to take care of inpatients with an intelligent system. Inpatients often take medicines on a regular basis. They also need to order

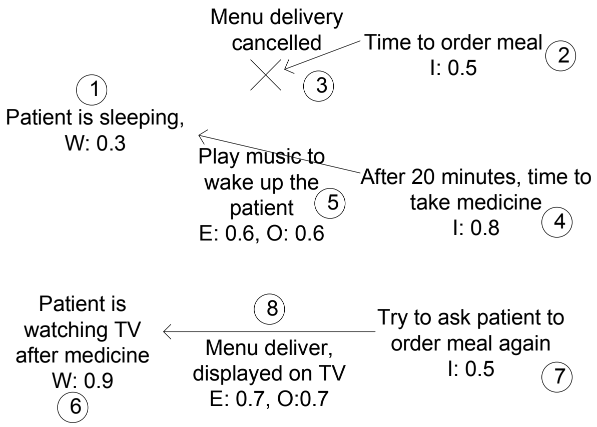


Figure 5.14: Inpatient care

meals two or three times a day. Usually, it is the nurses on shift who walk from one room to another to give medicines to patients or remind them to take pills, and bring menus for lunch or dinner. This is tedious. And the patient might be sleeping, that means the nurse will have to either wake her up or come back later.

With an intelligent system, we can improve the experience for both patients and nurses. Figure 5.14 illustrates such a scenario. When it is time for the patient to order a meal, the system detects that the patient is sleeping by using different sleep sensors to capture body position, limb movement, snoring, respiratory airflow, and so on. The system then assigns a low willingness of 0.3, which consequently aborts the meal-order interaction. After 20 minutes, the patient is supposed to take some pills according to the schedule. Since this is important, the system plays some soft music to wake up the patient. Then the system tries to ask the patient to select from the menu, and this time, this interaction takes place as the patient has a higher willingness.

In this way, the system takes care of these things on the nurses' behalf. Patients are interrupted only if the interaction is important and necessary. This saves some of the nurses' time and still provides a quality experience to patients.

There are many application scenarios in a hospital environment other than emergency handling and inpatient care. In the rest of this section, we discuss some other applications, such as patient scheduling in the context of the intelligent emergency system. The purpose is to show that the system can be leveraged to support different applications.

Patient scheduling is indispensable to medical professionals. This can be complicated when doctors are practicing in multiple clinics. Rescheduling also happens from time to

Doctor Adam's schedule	9 – 10am	10 – 11am	11am−5pm
Booking details		Bob's regular visit	Mike's operation
Willingness	1	0.3	0

Table 5.1: Doctor Adam's schedule

time. There exist online scheduling services and appointment-management tools. Often, these tools and services are replacements for paper-based appointment systems. When it comes to scheduling and rescheduling, human actions, such as phone calls, are still involved.

With the intelligent emergency system, it is possible to improve this aspect of patient scheduling. Each doctor has a calendar; each spot on the doctor's calendar is marked with a willingness value based on the booking. If a spot has not been booked, the willingness value will be high, otherwise the value will be low. When a patient tries to book an appointment, the system compares the importance of booking with the willingness value on the doctor's calendar to decide whether this booking request can be approved.

Table 5.1 shows that Doctor Adam is free before 10 o'clock in the morning, then he has a regular visit by Bob. Afterwards, his schedule is full for an operation for Mike. Suppose another patient Henry has an urgent need to see the doctor that day, and the visit will take more than one hour. In this case, if the importance of Henry's visit is greater than 0.7, the system will cancel Bob's appointment and Bob has to reschedule his original regular-visit appointment, otherwise Henry's request is denied, and no change is made to the doctor's schedule.

We can see that in this case, we only use the filter stage of the non-intrusive-computing model. The delivery stage is also applicable when the system needs to notify the doctor and the patient about the changes.

Existing hospitals often have color coded emergency systems. This section discusses an intelligent emergency system for Code Blue. Non-intrusive computing is applied in this system. If all the underlying context is available, we are able to build such a system that improves and facilitates handling Code Blue scenarios. Some applications are described to show some details of instantiating the model for non-intrusive computing. We think that this intelligent emergency system can also support other color-coded emergency scenarios, such as Code White.

This case study describes how non-intrusive computing may be used in a hospital environment, and what applications may be supported in this area. The aim is to show that non-intrusive computing provides general values to different domains, and its model and instantiations may improve scenarios where interactions are dealt with.

The next chapter concludes this thesis and discusses future work.

Chapter 6

Conclusions and future work

The goal of this thesis is to tackle intrusiveness in pervasive and ubiquitous computing. A model is proposed and prototyped. The solution is comprehensive in that it considers the intrusion of an interaction from both the receiver's and the other users' points of view. This work establishes a framework where problem abstraction and system construction are combined, and where new and further research is spawned. Section 6.1 concludes the thesis and Section 6.2 provides some ideas for future research.

6.1 Conclusions

The proliferation of devices with small form factors, such as sensors and actuators, and the trend of consumer objects becoming digital and "smart," such as smart phones and digital picture frames, indicate some progress towards Weiser's vision. However, in spite of many aspects that have been articulated, a pervasive-computing system has to reduce the distractions to users caused by various kinds of interactions.

An observation [49] conducted by a professor at the University of California at Irvine showed that in two high-tech companies, "each employee spent only 11 minutes on any given project before being interrupted and whisked off to do something else... and each time a worker was distracted from a task, it would take, on average, 25 minutes to return to that task." These results underline the importance of non-intrusive computing and motivate our research.

We extract a model from the problem domain. The model is further divided into two stages mimicking the lifespan of an interaction. The two stages suggest different but related questions. In the filter stage, the question to answer is whether an interaction is deliverable. In the delivery stage, it becomes whether a deliverable interaction will intrude on other people in the same location as the receiver. From the model, we propose a general solution. Importance and unwillingness are compared to determine deliverability, and effectiveness and overtness are introduced to ensure appropriate delivery.

This general solution places two requirements on how the system is implemented. The first one is that the importance and willingness, controlled by a sender and a receiver, have to be brought together for comparison. The second is that the middleware system has to provide appropriate context support so that context can be applied incrementally. More context results in better decisions made by the model.

Being able to model the intrusiveness and understand how to manage it is one of the major contributions. The model is then analyzed in order to understand its performance. Multiple approaches are tried. One of the results is that, in the hybrid dropping approach, if half of the successfully delivered interactions cause intrusion on other users due to inappropriate usage of target devices and notification methods, then the combined error rate is as high as 47%. With our system in place, this number could be reduced to zero, given accurate context. The user study demonstrates agreement on the the model and its parameters, and the results also show that the majority of the users agree on the parameter values. The average values of each parameter in different scenarios can be used as the default values in system instantiations.

Although the prototyping of the model is based on Jabber, the design of the system is a contribution in that it emphasizes the requirement for a communication bus and a naming mechanism for all participants. Context support is as important as the communication and naming support in the system, and is managed through the use of an ontology and a pubsub facility. Security is considered and designed into the system. Jabber is chosen because it already has many of the required features which we may reuse. However, these features are not enough in terms of system development. More importantly, they do not address the problems directly. We still need to develop components and integrate them together. The non-intrusive-computing system is the second major contribution that the thesis makes.

Jabber, as an extended IM system, goes beyond supporting non-intrusive computing. It can exchange many forms of structured data for many more uses than chatting. It is easy to use, with familiar interfaces. It is able to capture recurring themes from previous research on middleware. The standardization of Jabber makes this technology potentially pervasive. Although, this is not entirely related to non-intrusive computing, having a generic platform where a pervasive-computing ecology can be built will foster non-intrusive computing and its applications. For example, with a smart walker [14], we may extend the interaction management to elderly people in modern smart homes or health-care environments. We consider this a minor contribution of our work.

We use a quantitative approach together with the modeling. Atomic and complex modifiers are proposed. Context is used flexibly in both stages. Identifying public and private delivery candidates reduces the complexity of the selection process. These are also minor contributions of this dissertation.

6.2 Future work

We address the intrusion problem with a comparison model, which features a few parameters and selection criteria. This model is generic and can be applied to different domains (Section 5.4). However, it needs improvement in a number of respects.

Some interactions may occupy more of the receiver's time than others. For example, sending a document to the receiver and expecting him to read it immediately requires more time than asking the receiver to approve a purchase. At the moment, our model does not utilize the temporal properties of an interaction. When there are multiple concurrent interactions for a receiver, this ability is desirable, as we can use it to order interactions based on the receiver's schedule and the time available to him.

It is often the case that senders and receivers need to interact multiple times to solve a problem. These multiple interactions may be grouped together into a session. If one interaction of this group is deliverable, others are likely to be deliverable as well. Our model does not have a notion of session. In other words, it treats related interactions independently. In order to address this issue, one might add an additional layer that manages sessions, and apply deliverability to the session as a whole.

In Section 2.2, we mentioned that we only deal with interactions that have one sender and one receiver. The next step is to expand this to one-to-many interactions. One approach is to treat a one-to-many interaction as many one-to-one interactions, and to use the existing model to filter and deliver them sequentially. However, this may result in some interactions being delivered while others are not, depending on the willingness of individual receivers, which may or may not be appropriate. A different approach is to group receivers together, and to consider some aggregated "willingness." New questions will come up, such as how to aggregate the willingness of multiple receivers. We may introduce weights into the aggregation. For example, if a receiver with the highest role expresses enough willingness to accept an interaction, it may be reasonable to deliver that interaction in spite of some other receivers not being willing to interact. Furthermore, all receivers may not be in the same environment. If each receiver is in an isolated environment, we are still able to deliver the interaction with the existing effectiveness and overtness criteria, as there is no interference among different environments. However, if some of the receivers are in the same environment, the delivery becomes more complicated, because a delivery candidate may or may not be effective for multiple receivers. For example, if two receivers in the room are reading documents, a phone ringing is effective to both of them, but if one if them is listening to music, then ringing is less effective to him. Hence, the difficult part is to model and measure the effectiveness of delivery candidates for multiple receivers in the same space. Sophisticated context is likely needed in this situation.

These are the major directions in which our model may be stretched and improved. With regard to the intrusion problem generally, there could be other completely different approaches. Here, we discuss using an economic model and intelligent agents to manage interactions.

In an economic model, an interaction, such as a message, is associated with a price. Either the sender or the receiver pays the price to make the interaction happen. Devices may also have prices that have to be paid in order to use them. There are also utility functions, so that for a successful interaction, some participants receive rewards. How to assign roles, such as sellers, buyers, or market mediators to participants depends on the goals and the assumptions of the system. For example, in our case, we assume the sender knows the receiver, as interactions are reference-coupled (Section 2.2), and the goal is to reduce the intrusiveness to the receiver. Presumably, every interaction is intrusive in some sense, hence the sender should pay some price to purchase the receiver's attention and time in dealing with an interaction. The price of the interaction is then determined by how busy the receiver is. The busier the receiver is, the higher the price. Of course, the price is also influenced by other factors, such as the temporal properties of the interaction, assuming we expand our model as mentioned above. If it turns out that the sender sends a valuable message to the receiver, the sender receives rewards, which may be used towards the next time he wants to send a message.

When a message is delivered to some devices, especially if the devices are shared (such as a public whiteboard) or owned by an individual (such as a cell phone), either the sender or the receiver will have to pay the price required by the device for delivery. Public devices should be more expensive than private ones. This provides flexibility to the sender. If he thinks the message is very important, he could influence how the message is delivered by paying a larger price.

Intelligent agents could be used in combination with an economic model. Each agent represents a user and acts on his behalf based on his preferences. Agents could negotiate the price. For example, the sender's agent may tell the receiver's agent that the message is important and useful to the receiver, then the receiver's agent may agree on a lower price. Agents may also borrow money from each other. We mentioned that one future direction of our model is to address one-to-many interactions. The prices for such interactions tend to be higher. With an economic approach, the sender's agent may need to borrow money in order to pay the price. Agents may also cooperate to deliver interactions. Assuming that an agent has the right to use devices owned by its user, a multimedia interaction may be delivered to a screen and a stereo controlled by two different agents.

Although economic models are different from ours, they share some essential aspects, such as whether an interaction is important, how busy a user is, and so on. Comparing with our model, the economic one is more familiar to users as it is closer to our daily lives. At the same time, it involves more roles and procedures. Issues such as how to determine price and utility functions, who monitors the market, whether the overall amount of money should be constant and finite, and so on, should be investigated and figure in our future concerns.

There are other issues that we plan to investigate. One is to develop commutative modifiers for the importance and the willingness. Sometimes, when there is more than one modifier, we will need to consider the order of applying them. One way to make the modifiers commutative is to use exponential functions, since the importance and the willingness remain in $[0,1]$. However, it is not always easy to find appropriate exponential functions. Some modifiers take effect based on the original values, such as the importance modifier for roles in Section 2.3.6; other may yield new values irrespective of the old ones. For example, when the subject of an interaction matches the current activity of the receiver, it is reasonable to assume that the receiver will want to receive this interaction, so the new willingness is simply set to 1. We plan to devise a number of common modifiers and apply them in different orders and study their inter-relationships.

In Section 2.4.4, an assumption is made about the relationship between effectiveness and overtness of delivery candidates. However, this is an approximation (see the user study in Section 2.5), and may not always be accurate. For example, a cell-phone vibrating is a private delivery candidate, which ignores the fact that if a cell phone is vibrating on a meeting table, most of the attenders will notice it, which in turn invalidates the assumption that its overtness is negligible. This shows that the effectiveness and overtness of a delivery candidate are dynamic and current context can modify them. To address this, we need to use context to determine the effectiveness and overtness. This is difficult as there are many situations to consider. Besides, context may further change the effectiveness for the intended receiver. Taking the the same cell-phone example, if it vibrates on a table far away from the meeting, its overtness becomes negligible again, but at the same time, its effectiveness also becomes negligible. Obviously, we will have to make a few, but less restrictive, assumptions in order to tackle this problem. In addition, fuzzy logic may be applicable when studying the correlations among different context elements, which is beneficial not only to this problem, but also to that of calculating the combined effectiveness and overtness of multiple delivery candidates (see Section 2.4.2).

In terms of future prototyping and testing, we will keep developing the non-intrusive-computing system, such as deploying sensors to report the door status, based on which we can infer a user's willingness. We will also improve the user interface and release our system to other users.

Bibliography

[1] *Oxford English Dictionary*. Oxford University Press, second edition, March 1989. ISBN 978-0-19-861186-8. URL `http://www.oed.com/`. 3

[2] Aware Home Research Initiative. URL `http://awarehome.imtc.gatech.edu/`. 78

[3] Piotr D. Adamczyk and Brian P. Bailey. If not now, when?: the effects of interruption at different moments within task execution. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 271–278, New York, NY, USA, 2004. ACM. ISBN 1-58113-702-8. doi: http://doi.acm.org/10.1145/985692.985727. 9

[4] Piotr D. Adamczyk, Shamsi T. Iqbal, and Brian P. Bailey. A method, system, and tools for intelligent interruption management. In *TAMODIA '05: Proceedings of the 4th international workshop on Task models and diagrams*, pages 123–126, New York, NY, USA, 2005. ACM. ISBN 1-59593-220-8. doi: http://doi.acm.org/10.1145/1122935.1122959. 9

[5] Peter Andrews. Vying for your attention: Interruption management. Executive technology report G510-3939-00, Executive technology report, IBM Business Consulting Services, July 2004. 9

[6] Daniel Avrahami and Scott E. Hudson. Qna: Augmenting an instant messaging client to balance user responsiveness and performance. In *Proceeings of the ACM Conference on Computer Supported Cooperative Work (CSCW*, pages 515–518. ACM Press, 2004. 9

[7] Brian P. Bailey, Joseph A. Konstan, and John V. Carlis. Measuring the effects of interruptions on task performance in the user interface. In *In IEEE Conference*

on Systems, Man, and Cybernetics 2000 (SMC 2000, pages 757–762. IEEE Press, 2000. 9

[8] Francisco J. Ballesteros, Enrique Soriano, Gorka Guardiola, and Katia Leal. Plan B: An Operating System for Ubiquitous Computing Environments. In *Proceeding 4th annual IEEE International Conference Pervasive Computing and Communication*, pages 126–135, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2518-0. doi: http://dx.doi.org/10.1109/PERCOM.2006.43. 74, 75

[9] Francisco J. Ballesteros, Enrique Soriano, Gorka Guardiola, and Katia Leal. Plan B: Using Files instead of Middleware Abstractions. *IEEE Pervasive Computing*, 6(3):58–65, July-September 2007. ISSN 1536-1268. doi: http://doi.ieeecomputersociety.org/10.1109/MPRV.2007.65. 74, 75, 76

[10] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. Mobile-Agent Coordination Models for Internet Applications. *Computer*, 33(2):82–89, 2000. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/2.820044. 11

[11] D. Chamberlin. XQuery: An XML Query Language. *IBM Systems Journal*, 41(4):97–615, 2002. URL `http://researchweb.watson.ibm.com/journal/sj/414/chamberlin.html`. 99

[12] Ellick Chan, Jim Bresler, Jalal Al-Muhtadi, and Roy Campbell. Gaia Microserver: An Extendable Mobile Middleware Platform. In *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, pages 309–313, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2299-8. doi: http://dx.doi.org/10.1109/PERCOM.2005.22. 68

[13] Hao Chen and James P. Black. A quantitative approach to non-intrusive computing. In *The Fifth Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous'08)*, Dublin, Ireland, July 2008. 8

[14] Hao Chen, James P. Black, Omar Zia Khan, and Kamran Jamshaid. Data-centric support of a smart walker in a ubiquitous-computing environment. In *The 2nd International Workshop on systems and Networking Support for Healthcare and Assisted Living Environments (HealthNet'08)*, Breckenridge, Colorado, USA, June 2008. 125

[15] Harry Chen, Filip Perich, Tim Finin, and Anupam Joshi. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. In *International Conference on Mobile and Ubiquitous Systems: Networking and Services, Boston, MA.*, pages 258–267, Augest 2004. URL `http://csdl.computer.org/comp/proceedings/mobiquitous/2004/2208/00/22080258abs.htm`. 101, 114

[16] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A Context-based Infrastructure for Smart Environments. In *Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE '99)*, pages 114–128, Dublin, Ireland, December 1999. URL `http://www.cc.gatech.edu/fce/contexttoolkit/pubs/MANSE99.pdf`. 80

[17] T. Dierks and C. Allen. The TLS Protocol Version 1.0, January 1999. URL `http://www.ietf.org/rfc/rfc2246.txt`. 96

[18] Abdur-Rahman El-Sayed and James P. Black. Semantic-Based context-Aware Service Discovery in Pervasive-Computing Environments. In *First IEEE Int. Workshop on Services Integration in Pervasive Environments*, pages 9–14, Lyon, France, June 2006. 103

[19] David Garlan, Dan Siewiorek, Asim Smailagic, and Peter Steenkiste. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 1(2): 22–31, April-June 2002. ISSN 1536-1268. doi: http://dx.doi.org/10.1109/MPRV.2002.1012334. 8, 65

[20] Iván E. González, Jacob O. Wobbrock, Duen Horng Chau, Andrew Faulring, and Brad A. Myers. Eyes on the road, hands on the wheel: thumb-based interaction techniques for input on steering wheels. In *GI '07: Proceedings of Graphics Interface 2007*, pages 95–102, New York, NY, USA, 2007. ACM. ISBN 978-1-56881-337-0. doi: http://doi.acm.org.proxy.lib.uwaterloo.ca/10.1145/1268517.1268535. 9

[21] Robert Grimm. *System Support for Pervasive Applications*. PhD thesis, Department of Computer Science and Engineering, University of Washington, 2002. 71, 72

[22] Robert Grimm. One.world: Experiences with a Pervasive Computing Architecture.

IEEE Pervasive Computing, 3(3):22–30, July-September 2004. ISSN 1536-1268. doi: http://dx.doi.org/10.1109/MPRV.2004.1321024. 71, 73

[23] Edward S. De Guzman, Moushumi Sharmin, and Brian P. Bailey. Should i call now? understanding what context is considered when deciding whether to initiate remote communication via mobile devices. In *GI '07: Proceedings of Graphics Interface 2007*, pages 143–150, New York, NY, USA, 2007. ACM. ISBN 978-1-56881-337-0. doi: http://doi.acm.org/10.1145/1268517.1268542. 9

[24] Joanne L. Harbluk and Y. Ian Noy. The impact of cognitive disraction on driver visual behaviour and vehicle control. Technical Report TP 13889E, Transport Canada, February 2002. http://www.tc.gc.ca/roadsafety/tp/tp13889/menu.htm. 9

[25] G. R. Hayes, S. N. Patel, K. N. Truong, G. Iachello, J. A. Kientz, R. Farmer, and G. D. Abowd. The Personal Audio Loop: Designing a Ubiquitous Audio-Based Memory Aid. In *Mobile HCI*, volume 3160/2004, pages 168–179. Springer Berlin / Heidelberg, 2004. URL `http://springerlink.metapress.com/content/jm5x3c4h4wvtw1nb/?p=86ab3ae6e81d458fa2bc90ac5495388c&pi=14`. 81

[26] Hewlett-Packard. Jena - A Semantic Web Framework for Java. URL `http://jena.sourceforge.net/`. 100

[27] Insurance Information Institute. Cellphones and driving, June 2008. http://www.iii.org/media/hottopics/insurance/cellphones/. 9

[28] Jabber Software Foundation. Jabber.org: open instant messaging and presence. URL `http://www.jabber.org`. 6

[29] Brad Johanson and Armando Fox. The Event Heap: A Coordination Infrastructure for Interactive Workspaces. In *WMCSA '02: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, page 83, Washington, DC, USA, June 2002. IEEE Computer Society. ISBN 0-7695-1647-5. 70

[30] Brad Johanson, Armando Fox, and Terry Winograd. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. *IEEE Pervasive Computing*, 1(2):67–74, 2002. ISSN 1536-1268. doi: http://doi.ieeecomputersociety.org/10.1109/MPRV.2002.1012339. 69, 70

[31] Brad Johanson, Armando Fox, and Terry Winograd. The Stanford Interactive Workspaces Project. Technical Report 2004-05, Stanford University, August 2004. URL `http://hci.stanford.edu/cstr/reports/2004-05.pdf`. 69

[32] Tim Kindberg and John Barton. A Web-Based Nomadic Computing System. Technical Report HPL-2000-110, HP Laboratories Palo Alto, 2000. URL `http://www.hpl.hp.com/techreports/2000/HPL-2000-110.pdf`. 76

[33] Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Debbie Caswell, Philippe Debaty, Gita Gopal, Marcos Frid, Venky Krishnan, Howard Morris, John Schettino, Bill Serra, and Mirjana Spasojevic. People, Places, Things: Web Presence for the Real World. *Mobile Networks and Applications*, 7(5):365–376, October 2002. ISSN 1383-469X. doi: http://dx.doi.org/10.1023/A:1016591616731. 76

[34] Kevin A. Li, Patrick Baudisch, and Ken Hinckley. Blindsight: eyes-free access to mobile phones. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1389–1398, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-011-1. doi: http://doi.acm.org.proxy.lib.uwaterloo.ca/10.1145/1357054.1357273. 9

[35] Konrad Lorincz, David J. Malan, Thaddeus R.F. Fulford-Jones, Alan Nawoj, Antony Clavel, Victor Shnayder, Geoffrey Mainland, Matt Welsh, and Steve Moulton. Sensor networks for emergency response: Challenges and opportunities. *IEEE Pervasive Computing*, 3(4):16–23, 2004. ISSN 1536-1268. doi: http://doi.ieeecomputersociety.org/10.1109/MPRV.2004.18. 117

[36] J. Myers. RFC 2222 - Simple Authentication and Security Layer (SASL), October 1997. URL `http://www.faqs.org/rfcs/rfc2222.html`. 96

[37] Elizabeth D. Mynatt, Jim Rowan, Sarah Craighill, and Annie Jacobs. Digital family portraits: Supporting peace of mind for extended family members. In *CHI '01: Proceedings of the SIGCHI conference on human factors in computing systems*, pages 333–340, New York, NY, USA, 2001. ACM. ISBN 1-58113-327-8. doi: http://doi.acm.org/10.1145/365024.365126. 81

[38] Hewlett Packard. Cooltown. URL `http://www.hpl.hp.com/archive/cooltown/`. 76

[39] Rob Pike, Dave Presotto, Sean Dorward, Bob Flandrena, Ken Thompson, Howard Trickey, and Phil Winterbottom. Plan 9 from Bell Labs. *Computing Systems*, 8(3): 221–254, Summer 1995. 74

[40] Shankar Ponnekanti, Brian Lee, Armando Fox, Pat Hanrahan, and Terry Winograd. Icrafter: A service framework for ubiquitous computing environments. In *UbiComp '01: Proceedings of the 3rd international conference on Ubiquitous Computing*, pages 56–75, London, UK, 2001. Springer-Verlag. ISBN 3-540-42614-0. URL `http://springerlink.metapress.com/content/ty1v54d2bcyjkryj/?p=ce3026791b94458e8a84196c65880678&pi=6`. 70

[41] Sarvapali D. Ramchurn, Benjamin Deitch, Mark K. Thompson, David C. De Roure, Nicholas R. Jennings, and Michael Luck. Minimising Intrusiveness in Pervasive Computing Environments using Multi-Agent Negotiation. In *First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04)*, pages 22 – 26, Boston, Massachussets, USA, August 2004. doi: http://doi.ieeecomputersociety.org/10.1109/MOBIQ.2004.1331743. 10

[42] Donald A. Redelmeier and Robert J. Tibshirani. Association between cellular-telephone calls and motor vehicle collisions. *The New England Journal of Medicine*, 336(7):453–458, February 1997. http://dx.doi.org/10.1056/NEJM199702133360701. 9

[43] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. A Middleware Infrastructure for Active Spaces. *IEEE Pervasive Computing*, 1(4):74–83, October-December 2002. URL `http://ieeexplore.ieee.org/iel5/7756/25949/01158281.pdf?arnumber=1158281`. 8, 67

[44] Peter Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence, October 2004. URL `http://www.ietf.org/rfc/rfc3921.txt`. 97, 106

[45] Peter Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core, October 2004. URL `http://www.ietf.org/rfc/rfc3920.txt`. 92, 93, 102

[46] Mahadev Satyanarayanan. Mobile Information Access: Accessing information on demand at any location. *IEEE Personal Communications*, 3(1):26–33, 1996. 65

[47] Mike Schneider and Sara Kiesler. Calling while driving: effects of providing remote traffic context. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 561–569, New York, NY, USA, 2005. ACM. ISBN 1-58113-998-5. doi: http://doi.acm.org/10.1145/1054972.1055050. 9

[48] William Siler and James J. Buckley. *Fuzzy Expert Systems and Fuzzy Reasoning*. WILEY-InterScience, first edition, December 2004. ISBN 978-0-471-38859-3. URL `http://ca.wiley.com/WileyCDA/WileyTitle/productCd-0471388599.html`. 20

[49] Clive Thompson. Meet the Life Hackers. *The New York Times Magazine*, pages 40–45, October 2005. URL `http://www.nytimes.com/2005/10/16/magazine/16guru.html`. 123

[50] W3C. OWL Web Ontology Language Overview, February 2004. URL `http://www.w3.org/TR/owl-features/`. 100

[51] W3C. Resource Description Framework (RDF), 2004. URL `http://www.w3.org/RDF/`. 100

[52] W3C. SPARQL Query Language for RDF, January 2008. URL `http://www.w3.org/TR/rdf-sparql-query/`. 100

[53] W3C. SPARQL Query Results XML Format, January 2008. URL `http://www.w3.org/TR/rdf-sparql-XMLres/`. 101

[54] Zhenyu Wang and David Garlan. Task-Driven Computing. Technical Report CMU-CS-00-154, School of Computer Science, Carnegie Mellon University, May 2000. URL `http://reports-archive.adm.cs.cmu.edu/anon/2000/CMU-CS-00-154.ps`. 65

[55] Mark Weiser. The Computer for the 21st Century. *Scientific American*, pages 94–104, September 1991. 1

[56] Mark Weiser and John Seely Brown. The coming age of calm technolgy. *Beyond calculation: the next fifty years*, pages 75–85, 1997. 1

[57] Wikipedia. Bayesian network, February 2008. URL http://en.wikipedia.org/wiki/Bayesian_network. 31

[58] Wikipedia. List of countries by number of mobile phones in use, September 2008. http://en.wikipedia.org/wiki/List_of_countries_by_number_of_mobile_phones_in_use. 9

[59] Wikipedia. Dynamic programming, March 2008. URL http://en.wikipedia.org/wiki/Dynamic_programming. 41

[60] Wikipedia. Endowment effect, September 2008. http://en.wikipedia.org/wiki/Endowment_effect. 45

[61] Wikipedia. Fuzzy logic, February 2008. URL http://en.wikipedia.org/wiki/Fuzzy_logic. 15, 17

[62] Wikipedia. Knapsack problem, February 2008. URL http://en.wikipedia.org/wiki/Knapsack_problem. 41, 42

[63] Wikipedia. Likert scale, September 2008. http://en.wikipedia.org/wiki/Likert_scale. 44

[64] Wikipedia. Mobile phones and driving safety, September 2008. http://en.wikipedia.org/wiki/Mobile_phones_and_driving_safety. 9

[65] Dave Winer. XML-RPC Specification, June 1999. URL http://www.xmlrpc.com/spec. 94

[66] XMPP Standards Foundation (XSF). XMPP Extensions. URL http://www.xmpp.org/extensions/. 94