A Quantitative Approach to Non-Intrusive Computing

Hao Chen
University of Waterloo
200 University Avenue West
Waterloo, ON, Canada N2L 3G1
h8chen@uwaterloo.ca

James P. Black
University of Waterloo
200 University Avenue West
Waterloo, ON, Canada N2L 3G1
jpblack@uwaterloo.ca

ABSTRACT

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 applications, are involved in our daily lives, and users need to deal with them as well as with other people. People are easy to reach with multiple devices. We believe that there should be a systematic way to help users avoid intrusive interactions.

This paper analyzes the intrusion problem for generic interactions and presents 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? The essential factors in the model are quantified for comparison and computation. We also show how a non-intrusive-computing system can be implemented based on the model.

Categories and Subject Descriptors

I.6.5 [**Simulation and Modeling**]: Model Development; D.4.7 [**Operating Systems**]: Organization and Design—*distributed systems*

General Terms

Design, Management

Keywords

ubiquitous computing, pervasive computing, non-intrusive computing, context awareness

1. INTRODUCTION

In a ubiquitous-computing environment, saturated with computing devices and applications, users usually have multiple ways to communicate with each other and their surrounding environments, such as using cell phones, land-line phones, instant messaging (IM), and so on. Smart devices and software applications may also initiate interactions that involve users. Since users may be equipped with several communication devices, it is easy to reach them and make them the target of many interactions. If we do not provide support to help control these interactions, users may be disturbed often, and be unable to focus on their tasks. As an example, a chat request may result in a pop-up window, which not only blocks what a user is looking at, but also means that the user needs to deal with the interruption, either by disposing of the window, or by chatting with a friend on an unrelated topic. Studies [6] show that in high-tech companies, if a worker is distracted from a task, it takes an average of 25 minutes to return to that task. In addition, an interaction may intrude on both the receiver and people around. For example, a reporter's cell-phone ringing could interrupt a news conference and annoy the interviewee.

Current ubiquitous-computing research includes various aspects of this computing trend, however, dealing directly with intrusiveness seems to receive little attention. In this paper, we propose a model to address intrusive interactions in pervasive computing. The model considers intrusion on not only the receiver, but also on other users in the same physical environment. The model also aims at delivering interactions effectively by using appropriate devices.

The paper is organized as follows. First, we discuss previous work in Section 2. Then we describe our non-intrusive-computing model in Section 3. Sections 4 and 5 describe the architecture of a non-intrusive-computing system and its implementation. Then we conclude in Section 6.

2. PREVIOUS WORK

Much previous work tends to design the pervasive-computing environments to cooperate with users in an automatic fashion, so that users can have smooth and uninterrupted computing experiences, instead of having to worry about the details of low-level computing facilities and the complexity of environments. An example is the Aura project [3], 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 [5], which develops a context file system where relevant data can be mounted according to the current context of users. These systems focus on integrating components and reducing users' interventions in a ubiquitous-computing environment, but they do not address the intrusion problem of interactions directly.

A more relevant work is an agent-based approach to mini-

MobiQuitous 2008, July 21–25, 2008, Dublin, Ireland.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3473
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3473

mizing intrusiveness in a meeting environment [4]. It mainly concerns how to display a message when there is a meeting. If a public whiteboard is chosen, the agents negotiate with each other on behalf of their users based on the content of the message and users' interests, so that all users are comfortable with seeing the message on the whiteboard. However, this work does not address the problem of whether an incoming message is intrusive to the receiver.

Some existing systems such as electronic mail, telephone, and instant messaging are relevant to intrusiveness as well. 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. Fixed 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 intrusiveness has not been fully investigated. Existing systems deal with the problem in different ways but with various limitations. There is no systematic and common approach. Next, we describe our model and present our quantitative approach.

3. MODELING NON-INTRUSIVE COMPUTING

In general, interactions can be labeled as time- or reference-coupled or decoupled [1]. 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 her 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 an interaction is deliverable to the receiver. If it is not, it should be prevented, since it will intrude upon him. The second stage, given the interaction is deliverable, determines how to deliver it to the right devices and notify the receiver appropriately. We call the first part the filter stage, and the second the delivery stage, as illustrated in Figure 1.

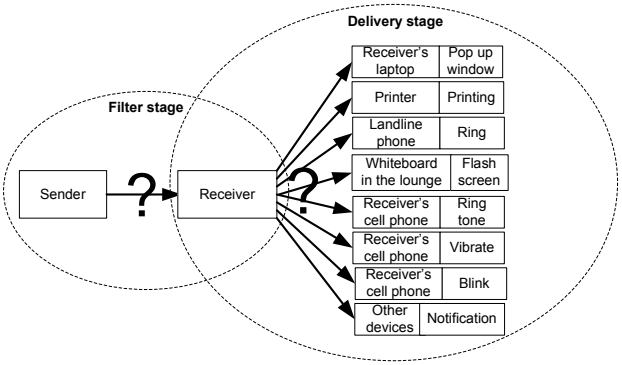

Figure 1: Overall modeling

The filter stage only involves a sender and a receiver, as we can see in Figure 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. A high importance usually indicates that this interaction should be delivered so that the receiver notices it, and a low willingness indicates that the receiver does not want to be disturbed in general.

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 1. We can see that on the receiver's side, in a pervasive-computing environment, there could be multiple devices available when an interaction happens, such as a laptop, a cell phone, or a whiteboard. Each of them may have one or more mechanisms for notifying users. For example, a cell phone can ring, vibrate, or blink. In order to choose one or more proper devices and corresponding notification methods, we 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 his 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

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3473
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3473

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 inappropriate devices or notification methods may cause intrusion on them. 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 appropriate destination devices in proper ways, so that receivers notice them and surrounding people are not disturbed. In the rest of Section 3, we discuss the details of modeling non-intrusive computing in two stages. In each stage, we analyze factors that affect intrusion and quantify them.

3.1 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 an interaction, $A \rightarrow B$.

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 compare them to deduce a value of deliverability.

3.1.1 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.

Note that 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.

3.1.2 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 the sender must know the purpose of the interaction, which makes him 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.

3.1.3 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.

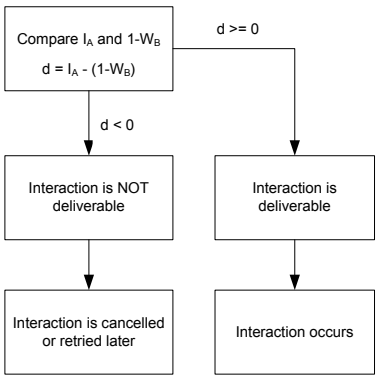

Figure 2: The comparison model

The model compares the importance I_A against the importance threshold, i.e., $1 - W_B$. 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. Non-deliverable interactions are discarded in the model. However, implementations will likely provide later or alternate delivery attempts or mechanisms, *i.e.*, changing the interaction to one that is not time- and reference- coupled. There is a boundary condition where $I_A = 1 - W_B$, in which case, we assume that this interaction is not intrusive.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3473
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3473

We also investigated a fuzzy-logic model for determining deliverability, in which there are fuzzy sets called *zero*, *low*, and *high* for importance and unwillingness, and *negative high*, *negative low*, *zero*, *positive low*, and *positive high* for deliverability. We then specify fuzzy-control rules based on these sets.

Since we are dealing with time-coupled interactions, for any such interaction, there are only two options, delivering or not delivering depending on whether the deliverability is greater than zero. For reasonable parameters and other choices made in implementing the fuzzy-logic controller, it turned out that the same decision was always made as in the much simpler direct comparison model, and so we did not pursue this further. Although fuzzy logic fits well into this scenario, as importance and willingness are fuzzy concepts, the fuzzy-logic controller acts identically to the comparison one.

We know that the sender and the receiver dictate the importance and the willingness respectively. However, this may not be 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. If a sender does not take this into consideration when specifying importance, receivers will be annoyed, either by receiving non-deliverable interactions or by not receiving deliverable ones. As another example, the willingness describes a general situation, but there are always exceptions that are unknown to the receiver. A person may have a low willingness, as he is busy. So this person is not interested in interactions in general, but only in related ones. All these situations can amount to saying that senders and receivers may not have necessary context. Section 3.1.4 discusses modifying raw importance and willingness using context.

3.1.4 Modifying importance and willingness

As an example of only one way of modifying importance, consider a failed student who tries to interact with an instructor, where the interaction is important to the student but not 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 interactions. 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 importance should be increased so that the comparison model does not otherwise block the interaction.

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, this particular modifier needs to increase or decrease importance according to the relative roles of the initiators and responders.

Assume roles are organized into a tree as in Figure 3. Given an interaction $A \rightarrow B$, the roles of A and B are R_A and R_B respectively. 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 arts faculties is 0. If A is higher than B in the role tree, the importance should be increased, otherwise decreased. The new importance should still be in $(0, 1]$. We propose the modifier in Algorithm 1.

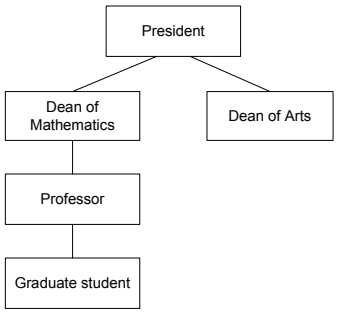

Figure 3: A partial role tree for a university

Algorithm 1 An importance modifier for roles

```
/*
 * given the original importance
 * and a role tree, this returns
 * the new modified importance
 *
 * R_A, R_B: integers represent A's
 *           and B's roles in the tree
 * R_A > R_B: if A is at a higher level
 *           than B
 */
if A or B is not on the role tree
  return I_A;
else if R_A > R_B,        /* increase I_A */
  newI_A ← I_A^(1/((R_A − R_B)+1));
else                      /* decrease I_A */
  newI_A ← I_A^((R_B − R_A)+1)
return newI_A;
```

Returning to the student-instructor example, the role distance between a student and an instructor is 1. Suppose the importance of the interaction is 0.8. It is high, as the student is desperate. Suppose the instructor's willingness is 0.3 (unwillingness 0.7), which means he is working on something and should only be disturbed for urgent interactions. Without the modifier, the instructor 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 unwillingness, which is also the importance threshold, 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 will make the intrusive interaction happen. We think that in a closed environment, users usually have some shared understanding of the importance of interactions. One can assume that users will not usually manipulate importance, as they know each other. Besides, at the system level, 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 completely block communication from selected users if they are aggressive.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3473
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3473

One might also wish to modify willingness. A user's willingness usually changes over time, so she can only specify a general willingness, instead of a specific one for everybody with whom she might interact. However, there are some interactions that will be blocked by the general willingness, but may actually be wanted by the receiver, and *vice versa*. 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 a meeting, Bob's secretary Alice tries to interact with him about some price information she just found for the equipment. If there is no willingness modifier, this interaction will not happen, because it is not an emergency, so it will not have a very high importance. The role relationship between Alice and Bob also reduces the importance. However, intuitively, Bob does want this interaction, as 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. However, there could be many such 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. The availability of the context information determines the usability of modifiers. More context can result in more modifiers becoming practical.

If there are multiple modifiers for importance or willingness, they may affect the willingness independently, i.e. they can both increase or decrease the willingness, or one could increase the willingness, while the other decreases it.

3.2 Delivery stage

Once the modified willingness and importance are compared, 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. 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 in Figure 4. 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 3.2.1. With feasible candidates, we apply the effectiveness and the overtness criteria (Sections 3.2.2 and 3.2.3) 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.

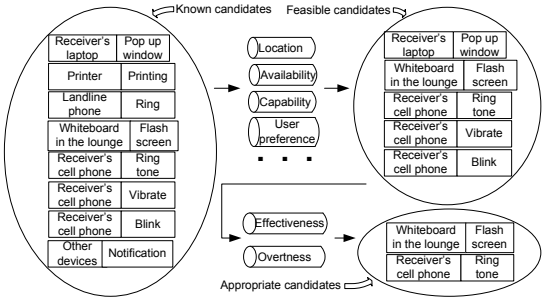

Figure 4: Delivery stage

3.2.1 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, there are a few other criteria that eliminate infeasible delivery candidates. For 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 4 lists four such criteria, each of which eliminates infeasible candidates 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 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 many 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 4 . 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 4. The next two sections discuss them in detail.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3473
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3473

3.2.2 Effectiveness

As we mentioned, a delivery candidate is a pair of a device and a notification method. Different candidates may have different levels of effectiveness when it comes to notifying users, as they use different mechanisms to capture a user's attention. 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.

We quantify effectiveness in $[0, 1]$. 0 means the device remains unchanged when an interaction is delivered, so the user receives no notification. 1 means that the notification always captures the user's attention. Effectiveness values also represent the probability of a user perceiving the notification. For a given candidate, its effectiveness is predefined and remains the same for all users. For example, cell-phone ringing and blinking might have effectivenesses of 0.6 and 0.4 respectively. Note that we exclude the situation where a delivery candidate could have different effectiveness on different users, such as ringing being less effective for people with hearing difficulties.

It is possible to combine more than one delivery candidate 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. From the probability point of view, the combined effectiveness measures the probability of the user perceiving the notification when using multiple candidates.

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, as well as the probability of a user perceiving it. Let c_1 and c_2 be two delivery candidates, $E(c_1 \cup c_2)$ the combined effectiveness of them, Based on probability theory, we have

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

$E(c_1 \cap c_2)$ is the joint effectiveness of c_1 and c_2. If we assume delivery candidates are independent in terms of capturing attention, then $E(c_1 \cap c_2) = E(c_1)E(c_2)$. 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.

The notion of the combined effectiveness can be expanded to a set of delivery candidates. Let $\sigma = \{c_1, c_2, ..., c_n\}$, then

$$E(\sigma) = E(c_1 \cup c_2 \cup ... \cup c_n) \in [0, 1]$$

Intuitively, if an interaction is important enough, or the receiver's willingness is high, we should make sure that the receiver does not miss this interaction. This means we should use an 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 from the filter stage.

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

$$SE = \{\sigma \subset S \mid E(\sigma) \geq d = I_A - (1 - W_B)\}$$

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 unwillingness (or the importance threshold), $1 - W_B$. In the context of effectiveness, the difference of I_A and $1 - W_B$ 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. The next section ensures our delivery choice based on the effectiveness criterion does not cause intrusion on other users.

3.2.3 Overtness

The discussion in the previous section is based on the observation that devices and notification methods imply different levels of effectiveness. Similarly, each delivery candidate has a degree of overtness. For example, a cell-phone ringing has a higher overtness than vibration. As another example, popping up a window on a big whiteboard has an even higher overtness than a cell-phone ringing in a class. We let overtness values of delivery candidates range from 0 to 1. To illustrate this, we might assign 0.8, 0.6, and 0 to white-board displaying, cell-phone ringing, and vibration respectively.

Let $c = (D, N)$ be a delivery candidate, then $O(c) \in [0, 1]$ is the overtness of c, and also the probability that some other users will be aware of the delivery through c. We also expand the overtness notion from a single candidate to a set of candidates, as in Section 3.2.2. Given $\sigma = \{c_1, c_2, ..., c_n\}$, then $O(\sigma) = O(c_1 \cup c_2 \cup ... \cup c_n) \in [0, 1]$. The assumption of two delivery candidates being independent as well as the related discussion on fuzzy logic in Section 3.2.2 are also applicable to 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, or a lunch-break situation has a loose overtness requirement where everybody can make more noise. We use OT (overtness threshold) to denote the requirement imposed by the environment. The range of OT is $[0, 1]$. An example assignment of OT to a meeting and a lunch break would be 0.1 and 0.9. These values are predetermined based on the environment context. A lower OT means a more strict overtness requirement. When OT becomes 0, it means right now there should be no overtness at all.

We define the following *overtness criterion*.

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

It says that we should select σ, a subset of the set S of all feasible delivery candidates, such that the overtness of σ is less than the current overtness threshold, OT.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3473
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3473

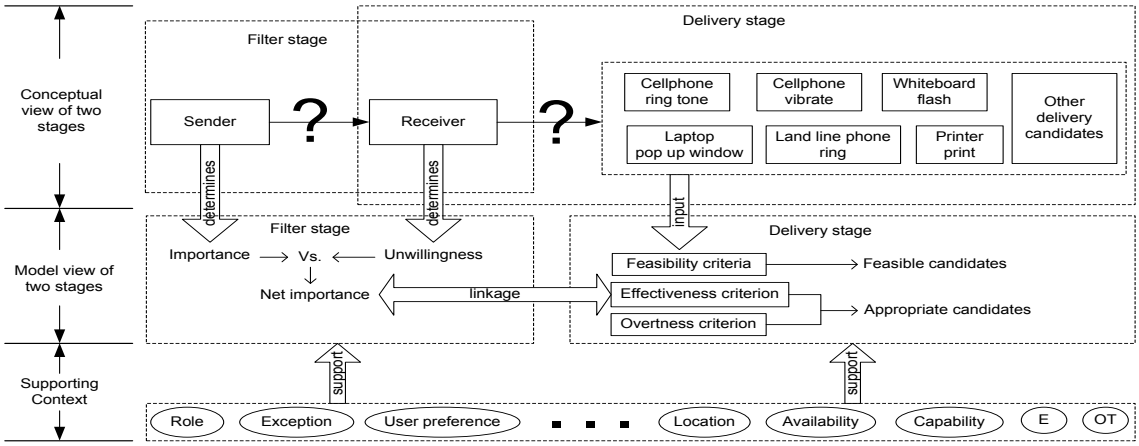

Figure 5: Different views of non-intrusive computing

We have discussed selection criteria for feasibility, effectiveness, and overtness. The next section describes how to perform the selection and summarizes the modeling of non-intrusive computing.

3.2.4 *Delivery-candidate selection*

The aim of the delivery stage is to select proper candidates. As shown in Figure 4, the feasibility criteria are applied first, then the effectiveness and overtness ones. Applying the feasibility criteria is straightforward given the necessary context. At the model level, we simply use context. Then at the system level (see Section 4), we describe how we manage it and discuss its representation and access mechanism.

The result of the feasibility criteria is the feasible set, FS. Conceptually, when the effectiveness and overtness criteria are applied to FS, they produce SE and SO; then their intersection, RE, is the final result. Each member of RE is a delivery choice (a set of candidates). For the delivery, we only need one such choice.

$$\begin{aligned} SE &= \{\sigma \subset FS \mid E(\sigma) \geq d = I_A - (1 - W_B)\} \\ SO &= \{\sigma \subset FS \mid O(\sigma) \leq OT\} \\ RE &= SE \cap SO \end{aligned}$$

Applying the effectiveness and overtness criteria to FS requires the combined overtness of the result to be less than OT, and the combined effectiveness of that result to be greater than the net importance. This problem is essentially a knapsack problem [12], where there is a set of items, each of which has a value and a cost. The knapsack 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 analogous to the capacity of the knapsack. The effectiveness of each candidate is the value of each item, and the combined overtness of the result is analogous to the total cost of selected items.

The difference is that in the knapsack problem, the cost or the value of multiple items is the sum of individuals, whereas in this candidate-selection problem, the effectiveness or overtness of multiple candidates is the combined probability of individuals. Also, the selection problem has a lower bound on the effectiveness, the net importance, while the knapsack one does not. Despite these differences, the selection problem is a $0-1$ knapsack problem, which is NP-complete [12]. Dynamic programming [11] can be used to solve these problems [12]. However, it requires much space to compensate for the run-time complexity, which may be a constraint on mobile devices if that is where the selection takes place. Since the number of feasible candidates in an environment is limited, say less than 20, it is possible to use exhaustive search.

In addition, there is a relationship between effectiveness and overtness that can be exploited to simplify the problem. For example, a whiteboard in a meeting environment has the same effectiveness and overtness, since everybody sees it. For another example, a cell-phone vibration has a high effectiveness and little or even negligible overtness. We are able to use these relationships to simplify the selection problem in many situations. However, we do not discuss these details here.

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 destination devices. Figure 5 summarizes the modeling of non-intrusive computing.

4. SYSTEM ARCHITECTURE

Section 3 describes a model for non-intrusive computing. We need to develop a system to reflect the model. We first articulate a few system requirements implied by the model.

- The system has to support time- and reference-coupled communications, and in multiple formats, so that it can take advantage of the generic nature of the model in this regard.
- Importance and willingness, although determined by separate entities, should be brought together for comparison.
- The system should manage context and provide flexible access to it.
- It should also integrate devices so that interactions can be delivered.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3473
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3473

Besides these requirements, in order to make the system practical, there are some general system issues, such as security, user interface, and so on.

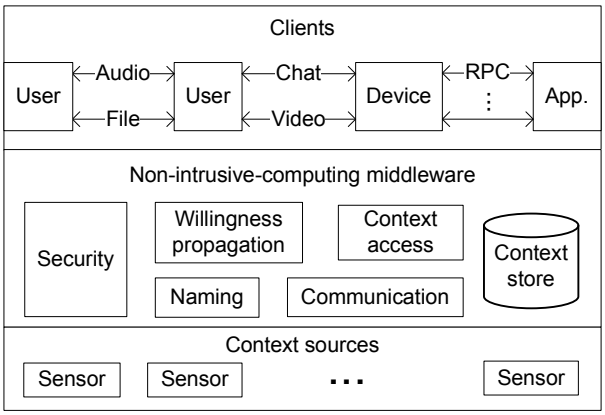

Figure 6: System architecture

Given these system requirements, our non-intrusive-computing system has an architecture illustrated in Figure 6. At the bottom level, sensors provide context, which is needed in both stages of the model, as summarized in Figure 5. Then there is middleware for non-intrusive computing. The core of the middleware is the communication and naming modules, which provides a communication platform to support interactions between any two entities within the system.

Sensor data is collected and aggregated into the context store. The middleware offers a flexible mechanism for users and applications to access context. A client may "pull" context by querying ontologies; the system may also "push" context to clients when it becomes available.

Willingness propagation addresses the filter stage in our model. It is where the importance and willingness meet and are compared with each other.

The security module protects the system at different levels. At the lower level, it ensures secure communication, for example, using SSL, among different system components. It also authenticates clients to the system. At a higher level, it provides access control for clients. Each client is able to specify a whitelist or blacklist of entities for communication.

The non-intrusive computing model does not constrain the format of communication. Thus on top of the middleware, clients may interact with each other in different ways. The communication module only establishes connections for two clients and provides transport support. It is the clients themselves who are responsible for handling the semantics of interactions.

The implementation of this system is influenced by existing technologies and software components. By reusing them, we need not build the system from scratch. The next section discusses the implementation details.

5. IMPLEMENTATION

We choose Jabber as the communication platform, a semantic-web ontology as the context representation format, and a publish/subscribe (pubsub) facility as a way to access context.

Jabber (`www.jabber.org`), based on the Extensible Messaging and Presence Protocol (XMPP), is a standardized IM system that enables the exchange of messages and presence information in near real time. Jabber has been extended with many more communication mechanisms than instant messaging (see `www.xmpp.org/extensions/`). To name a few, Jabber supports ad-hoc commands, RPCs, audio/video conversations, SOAP over XMPP, and so on. Jabber entities include human users, devices, and software applications. They all use XMPP to communicate with each other. The data exchanged in Jabber is in XML.

As an IM system, Jabber provides a presence model, where a user can specify his presence such as "away" or "do not disturb," and allow or deny subscription requests from other users. Once there is a presence change, the Jabber server notifies all subscribers. Jabber also extends traditional presence to include additional attributes such as the user's location, activities, and mood. In our case, we add willingness as part of the presence, so that the sender receives it as a presence subscriber and can compare it with the importance.

Some other features of Jabber are also important. It has a multi-tier security model; it provides server-side storage for clients that may be used to save user-specific information, such as preferences.

When modeling non-intrusive computing, we mention that in the filter stage, importance and willingness need to be modified so that they are more appropriate, and that in the delivery stage, there are a number of feasibility criteria that eliminate infeasible candidates. Both procedures require context as summarized in Figure 5, and the use of more and better context can yield better results.

The Web Ontology Language (OWL) [7] provides a standard mechanism for representing context. The underlying data model of OWL is the Resource Description Framework (RDF) [8]. RDF describes knowledge using triples of a subject, a predicate, and an object. The object in one triple may be the subject of another. 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.

SPARQL [9], also standardized by W3C, is a query language for RDF data. It uses a syntax similar to SQL. The following query example returns all devices in room DC3552D. SPARQL allows multiple `where` clauses to form a complex query.

```
select ?device
from devices.owl
where
{
  ?device subsumedBy "DC3552D"
}
```

With OWL and SPARQL, programs can retrieve desired context by issuing appropriate queries. However, in a ubiquitous-computing environment, some context changes often, such as the number of users in a room, the queue length of a printer, and so on. It is more efficient to push the changed context to applications than to require them to request it every time. We use pubsub for this purpose. If the object of an RDF triple is dynamic, then a URI is placed in the RDF triple, pointing to the corresponding node in a pubsub facility where the actual data can be obtained.

Our implementation has two major parts. One is a context manager that provides context support for the non-intrusive-computing model; the other embeds the model into

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3473
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3473

a Jabber client. For the context manager, we developed iContext, a Jabber server component. The ontology scheme is adopted and extended from SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications) ontologies [2]. iContext accepts SPARQL queries from clients. Results are returned in the XML format defined by [10]. The pubsub facility uses Openfire (`www.igniterealtime.org/projects/openfire/index.jsp`), an open-source Jabber/XMPP server.

We then modified Spark, a Jabber client from the same company as Openfire. For the filter stage, we add importance and willingness at the sender and the receiver sides respectively. Willingness is treated as part of the presence.

Figure 7 shows screenshots of a modified Spark Jabber client. The upper picture shows that the receiver is setting his willingness from an editable combo-box. Then this willingness is propagated to the sender (the middle part in Figure 7), because the sender subscribes to the receiver's presence.

The bottom part shows the sender composing a message with importance 0.5. The message is dropped if the importance is less than the unwillingness. Otherwise, the message is sent to the receiver. The net importance is appended at the end the message and also sent to the receiver, as the effectiveness criterion in the delivery stage needs it.

Note that it is inconvenient to always set willingness manually. We developed an algorithm to obtain willingness values. In the "set status message" window, there is a check box to "infer willingness automatically." Once selected, we check the receiver's Google calendar, door status, and IM presence, each of which is mapped to some value in $[0, 1]$. If the calendar entry is free, the value is 0, and 1 otherwise. The door angle measurement can be normalized to range over $[0, 1]$, and discrete values can be assigned to different presence settings. Then the three values are averaged as the willingness. This mechanism may not be always accurate, however, it serves as an example to illustrate possible automation using context. For the importance, the sender will set it for individual interactions, similar to the priority level in electronic mails. This is just one way to set willingness and importance. We imagine that different applications and systems will use different algorithms or inference mechanisms to determine the values for importance and willingness.

For the delivery stage, when the modified Spark client receives an interaction, it sends SPARQL queries to iContext. A simplified version of such a query is the following.

```
select ?d ?effectiveness ?overtness
from [...]
where
{
  ?d EFFECTIVENESS ?effectiveness .
  ?d OVERTNESS ?overtness .
  ?d subsumedBy "DC3552D" .
  ?d AVAILABLE "true"
}
```

The query retrieves the effectiveness and overtness of delivery candidates that are in `DC3552D` and available currently. We then use exhaustive search to determine the appropriate delivery candidates and forward the interaction. Next, we illustrate how a non-intrusive-computing system works.

When the system starts up, the static context of each device is placed into the ontology by the system administrator, and the dynamic context is updated by the Jabber wrappers

Figure 7: Filter-stage implementation on Spark

of devices.

The receiver reflects her willingness explicitly or the client software infers it automatically. The sender will obtain it through Jabber presence after subscribing to the receiver's presence. Thus the sender's Jabber client can compare the importance of the interaction with the importance threshold when it attempts an interaction. If it decides that the interaction, such as a message, is deliverable, it appends the net importance to the body element of the message's XML stanza and delivers the message to the receiver's client. Stage 1 in Figure 8 shows this process.

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. iContext executes the queries and returns XML results. The receiver client is then able to apply the effectiveness and overtness criteria on feasible candidates. The last step is to forward the message without the net importance to the selected delivery candi-

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3473
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3473

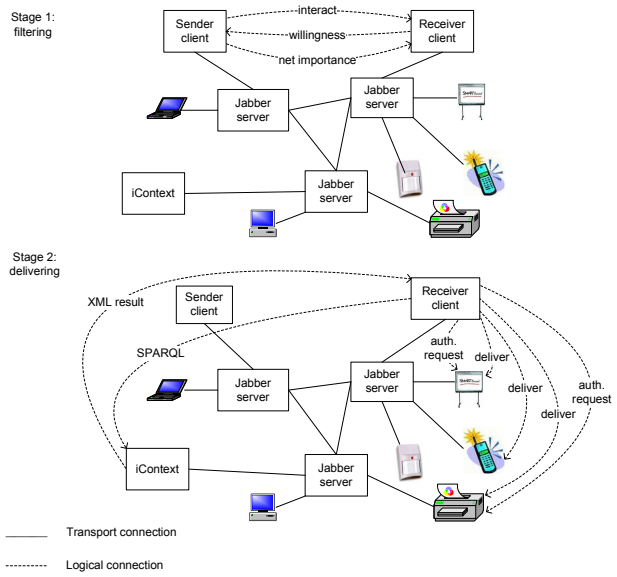

Figure 8: System illustration

dates.

Some of the devices in this scenario are public, such as the whiteboard and the printer in Figure 8. To prevent users, such as newcomers, from abusing them, and also increase security, we implemented a light-weight authentication mechanism, where the receiver client has to be authenticated before it can deliver any interaction to those devices. With this mechanism, 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 communications are 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 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.

6. CONCLUSIONS

In pervasive computing, users can be disturbed by unnecessary interactions. The major contribution of this paper is a model to address the intrusion problem for the receiver and other users. The model involves four essential factors in two stages: importance and willingness in the filter stage and effectiveness and overtness in the delivery stage. We quantify these factors and link the two stages through the net importance.

We choose existing technologies and frameworks in building our system so that we can leverage their features and benefit from their standards.

For future work, we should expand to one-to-many interactions. A natural approach is to treat a one-to-many interaction as many one-to-one interactions and 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. A different approach is to group receivers together, and consider the aggregated willingness.

Although our system has been used mainly by developers, we will also deploy it for others, and investigate such issues as how many intrusive interactions can be filtered, how effective the chosen delivery candidates are, what is an appropriate overtness threshold, and so on.

7. REFERENCES

[1] G. Cabri, L. Leonardi, and F. Zambonelli. Mobile-Agent Coordination Models for Internet Applications. *Computer*, 33(2):82–89, 2000.

[2] H. Chen, F. Perich, T. Finin, and A. 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, August 2004.

[3] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 1(2):22–31, April-June 2002.

[4] S. D. Ramchurn, B. Deitch, M. K. Thompson, D. C. D. Roure, N. R. Jennings, and M. 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.

[5] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. A Middleware Infrastructure for Active Spaces. *IEEE Pervasive Computing*, 1(4):74–83, October-December 2002.

[6] C. Thompson. Meet the Life Hackers. *The New York Times Magazine*, pages 40–45, October 2005.

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

[8] W3C. Resource Description Framework (RDF), 2004. http://www.w3.org/RDF/.

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

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

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

[12] Wikipedia. Knapsack problem, February 2008. http://en.wikipedia.org/wiki/Knapsack_problem.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3473
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3473