A Widget Based Architecture for Perceptive Presence

by

Frank R. Bentley

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2003

Author .....
Department of Electrical Engineering and Computer Science
December 20, 2002

Certified by .....
Trevor J. Darrell
Assistant Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

A Widget Based Architecture for Perceptive Presence

by

Frank R. Bentley

Submitted to the Department of Electrical Engineering and Computer Science
on December 20, 2002, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In order to better create systems that use presence and awareness information, it is possible to create a general computer vision interface (identifying such concepts as blobs, position, and velocity, etc.) and set of widgets (identifying concepts such as Person Present, Spatial Selection, Gaze Selection, etc.) to accurately determine information about a person's activity. A general interface can interface multiple computer vision systems to a layered stack of presence widgets that applications can rely on to obtain information about a given area. One such application, the Perceptive Presence Lamp, is currently running in the MIT AI Lab – linking two people together and allowing each one to see when the other is in, out, busy, or wanting to get their attention.

Thesis Supervisor: Trevor J. Darrell
Title: Assistant Professor

Acknowledgments

I would like to thank my parents for always providing encouragement to persue my dreams, and for finding a way for me to attend MIT. I would also like to thank Laura, my friends, and my housemates for all their encouragement while I was working many late hours to finish this work.

For his support of my research and the many comments he has provided, I would like to thank Prof. Trevor Darrell. Finally, this thesis owes a great deal to the help of Konrad Tollmar, his ideas, and his previous work.

This project was funded through MIT Project Oxygen [13].

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Ever since computers were first networked together, people have been using them for communication. From email, one of the first network applications, to the original MIT Zephyr Notification System [14] to the now-popular AOL Instant Messenger System [1], it has been the desire of many users to use computers to break down physical distances and provide a sense of awareness about another person in another place.

Beyond basic text interaction, other researchers have worked on bringing people together through sharing snapshots or video of users in remote places, such as the Portholes project at Xerox PARC [17]. These systems used video, or still shots, but required the human to do all the intelligent processing to determine if people were present, busy, or available to talk.

As interaction with computers moves away from the traditional domain of a computer and mouse moving a pointer on a screen, it is now possible for the computer itself to determine a person's state (including basic information like location and head pose, but also more useful information such as "present" or "busy"). Applications that can perform these functions are called Perceptive Presence applications. Perceptive Presence application are awareness tools that use input from any of a variety of sources (vision, sensors, etc.) and provide information about the people or activities in a space. These Perceptive Presence applications provide an alternative to the traditional screen-based GUI by using physical presence and gestures to replace the traditional keyboard and mouse. For example, Perceptive Presence applications

can share information about how many people are present in a area, where they are located, how active they are, or where they are looking. This information can then be used for communication applications, interacting with "objects" in a room by look or gesture (such as light switches, telephones, or selections on a screen), or for security purposes.

In this thesis, I present a general XML-based [2] computer vision interface for Perceptive Presence applications and a set of "widgets" that use this interface to infer activity information about a place. The interface connects to any number of computer vision modules. The architecture of this implementation is shown in Figure 1-1. This discussion will be grounded in an application, the "Perceptive Presence Lamp," [10] that is currently functional at the MIT Artificial Intelligence Lab using this architecture. The lamp is a device that conveys state (in office, out of office, interacting with lamp, and busy) between a pair of users in order to facilitate communication and awareness. The state is represented in a lamp on each user's desk that changes color to represent the state of the other person.

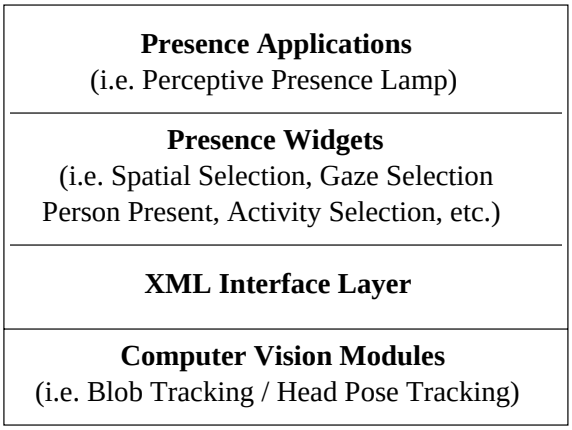

Figure 1-1: The architecture of a Perceptive Presence System. Applications can use information from Widgets such as if a person is present (and where they are), or where a person is looking. The widgets compute this perceptive information from data they get through an Interface Layer from Computer Vision Modules.

1.1 Perceptive Presence Applications

Perceptive Presence applications embody a shift away from traditional modes of interaction with a computer. These applications are based on Perceptual User Interfaces (PUIs) where user input is based on movement, location, or gesture. They seek to provide a sense of awareness about another space. Imagine the following scenario:

Consider two co-workers, John and Mary, who communicate frequently throughout the course of a day. John's office is located on the other side of a large office building and often Mary walks down to find that John is not present. Mary does not like to telephone John each time she wants to ask him something so that she doesn't interrupt his train of thought or call in the middle of a busy meeting.

Wouldn't it be great if Mary had a way to be aware of when John was free, or when he had guests? This is where presence applications come in.

1.1.1 Presence and Awareness

It is important to obtain a clear understanding of the terms "presence" and "awareness" in order to proceed further. There have been many papers written on both of these topics and each provides a slightly different definition. The following definitions provide detailed classifications of the terms and cite papers that seek to summarize research in the field.

Presence

The term presence in its current interpretation can be traced back to Marvin Minsky in 1980. Minsky used the terms "telepresence" and "remote presence" to describe a system that allowed a user to "see and feel" a remote location. [25] The term continued to be used over the years to describe various computer simulations as well as television, radio, and other media. In 1997, Lombard and Ditton performed an extensive survey of presence. [23] [24] Their results showed six main dimensions of presence. They are:

1. **Social Richness** - The degree of social warmth or intimacy conducted by the medium. Does the medium invite interaction?

2. **Realism** - In realism, their focus was on the perceived reality of the experience provided by the system. In other words, if the experience of the user was similar to a truly present experience.

3. **Transportation** - Transportation signifies the ability for someone to feel that they are in an environment and to suspend their disbelief so that they can interact in this created space.

4. **Immersion** - Immersion has to do with the feelings coming from a space. Does it feel like a person is involved and engrossed in this space?

5. **Social Actor within Medium** - This dimension has to do with the creating of inappropriate/illogical responses to a medium, such as talking to your television set.

6. **Medium as Social Actor** - Finally, the medium as a social actor refers to users thinking that the medium itself is in some way able to interact, or in some way human-like.

In my system, I try to create a sense of Social Richness and Realism. I hope that the user will feel like they are interacting directly with the user on the other side of the system (realism) and that it gets the user to interact as they would an office-mate by responding to signals about presence (social richness).

Awareness

Steinfield, Jang, and Pfaff split awareness into five distinct groups. Each group touches on a different aspect of awareness. [32]

1. **Activity Awareness** - "Knowing what actions others are taking at any given moment" This type of awareness is especially important in CSCW applications where multiple people are working together to produce a document or piece of

code. Knowing what files others are viewing or editing for example would be examples of Activity Awareness.

2. **Availability Awareness** - "Availability of people" and "social awareness" Availability Awareness is important in knowing whether people are available to talk. This can be important to support distance communication and collaboration where it would be impossible to drop in on a colleague to talk, but the individual would not wish to interrupt someone while in a meeting or otherwise busy. Some systems provide information similar to what one would see walking by an open door. (Examples: AIM [1], ICQ, EuroPARC's RAVE [19])

3. **Process Awareness** - "Gives people a sense of where their pieces fit into the whole picture, what the next step is, and what needs to be done to move the process along" This form of awareness is useful in applications designed for large group use and for managing large projects and was the key to much Computer Supported Collaborative Work (CSCW) projects in the 1990s.

4. **Perspective Awareness** - "Giving group members information helpful for making sense of others' actions, such as background on team member beliefs and knowledge" Perspective awareness allows people to gain information about others' experience. Individuals can provide information on their own work and back ground so that others can seek their advice or know which tasks to assign to the m. This can work as an enhanced phone book type system, or as a part of a larger system.

5. **Environmental Awareness** - "Focuses on events occurring outside of the immediate workspace that may have implications for group activity." Environmental awareness is important for applications that need to provide external information such as news or economic conditions that could impact people's decisions.

In this work, I chose to focus on the area of Availability Awareness, as I I feel this is the most relevant in conveying information on presence. Knowing when a person

is in, or when they are busy (for example in a meeting) can convey their state for the purposes of communication. The notion of Activity Awareness is closely related to Availability awareness and could be added to the current system by adding knowledge of which user locations and motions correspond to a given activity (as in [12]).

1.2 The Perceptive Presence Lamp

The Perceptive Presence Lamp (Figure 1-2) is an example of a Perceptive Presence Application. The lamp serves to connect two physically separated places by signaling information about the activity at the other location. This provides a sense of awareness and connection between the two locations and allows others to know when someone is free, or just get a reassurance that a person is where they are expected to be. The hope is that this connection will stimulate communication or collaboration and allow people to better be in tune with the schedules of those around them.

The idea of choosing a lamp, was to find something that fit into the natural desktop and did not feel to out of place. Also, as Tollmar and Persson point out, the concept of a light is a natural way to think about presence. [34] People use lights everyday to signal when they are home, or away. Many people even leave lights on to make people think they are home when they are really away. Thus it seems natural to have a light be a representation of someone's remote presence. Also, the idea of integrating awareness information with furniture in the workplace can be seen in IBM's BlueSpace project. [3]

1.2.1 Setup

At each location, a lamp and a stereo camera are placed on a desk/workspace. There are two versions of the lamp. The original prototype, described in this section, uses a dimmable lightbulb and varies intensity to indicate different states. The current model uses a color changing lamp and changes the color of the lamp to signal to the user what is happening in the other space. Table 1.1 illustrates the states that the lamp can be in.

Figure 1-2: The usage of the Perceptive Presence Lamp. In the top row, both users are working, and their lamps are dim. In the bottom row, the user on the right has looked at his lamp (much as a person would look at an office mate to get their attention) thus brightening the lamp at the other location. This brightening got the user on the left's attention and he returned the glance to the user on the right.

State	**Dimmable Lamp Action**	**Color Changing Lamp Action**
A person is present at their desk	dim	green
There are multiple people present – the person is busy	off	red
The person is interacting with his lamp – he is trying to get the user's attention	bright	blue
No one is present	off	off

Table 1.1: States of the Perceptive Presence Lamp

The lamp uses the concept of "glances" to aid communication. If a user looks at his lamp in our first prototype, much like he would look at an office mate to get their attention, this is a signal of a desire to communicate, and the lamp in the other location brightens.

1.2.2 Usage Scenario

Let us go back to John and Mary.

With the Perceptive Presence Lamp, Mary can now look at the state of her lamp to know if John is present. If the lamp was off, she would know that John is not around, and not to bother walking down. If the lamp indicated that there were multiple people present, she would also not walk down because she would not want to interrupt this meeting. However, if the lamp indicated that he was present by himself at his desk, she could send him a glance through the lamp. If he returned this glance, she would know that it was a good time to walk down. If he didn't, she could assume that he was actively involved in something so much that he didn't notice the lamp change or that he just didn't have time right now to work with Mary.

The lamp would help John and Mary to collaborate more effectively and to respect each others time without unnecessary interruptions.

1.2.3 Initial Prototype

Before creating the Widget interface system, I built a prototype of this lamp using existing technologies (see Figure 1-3) in order to better understand the requirements of a Perceptive Presence application. The initial prototype used an X10 [37] Motion Sensor to determine if a person was present and a monocular camera running a face detector to find frontal faces in a specified window of interest.

A series of user tests with the lamp indicated that the concept was useful but that the implementation was not always reliable in its estimations of the activity. For example, wind blowing a curtain or plant could set off the motion detector. Also, people present in the room could set off the face detector by looking at the lamp

Figure 1-3: The original Perceptive Presence Lamp prototype. This lamp used an X10 motion sensor to determine if a person was present and a camera running a face detector in a subset of the field of view to determine if a person was trying to send a glance using the lamp.

from a certain angle. After testing different areas of interest for the face detector (the whole field of view, a strip along the left side, a similar strip along the right side, and a circle emanating from the center of the image) it was determined that a new approach would be necessary. From there, I started work on the Interface System and Widgets described in later chapters.

Chapter 2

Previous Work

My work in creating this system builds upon the work of many different areas. Most directly, it uses the concept of a perceptual user interface driven by body actions and not interactions with a traditional keyboard and mouse. It also builds upon the work of communications systems and Media Spaces that used text, video, and audio to show people what was going on in another place, or simply let someone tell the user what was going on. Finally, it builds upon work in the area of computer vision in creating an interface to computer vision modules for Perceptive Presence.

2.1 Context Aware Spaces

In the past five years, many research groups have built intelligent spaces, or environments that are "context aware" by knowing the activities occurring within them. In general, these intelligent spaces combine multiple modes of sensors and are based on agent or widget oriented frameworks where the agents or widgets cooperate with each other to share state about the environment and act appropriately. [4] [30] [5] [16]

In his PhD thesis at Georgia Tech [16], Dey describes a Context Toolkit made up widgets that take input from sensors and provide information ("context") about what is going on in the environment. The Aware Home project at Georgia Tech [15] builds a shared widget set from this concept that focuses on providing perceptual information. The widgets in our system are much in the spirit of these context widgets and provide

many of the same functions such as determining if a person is present or moving; however, our widgets go further in providing a general interface and application-level GUI-like tools such as selection based in computer vision techniques.

Another Intelligent Environment that helped inspire this work is the Intelligent Room Project at the MIT Artificial Intelligence Lab and its agent-based programming language Metaglue. [4] [30] In the Intelligent Room, agents receive information about activities going on in the room, and can send this information to other agents so that they can act on this information. The sort of publish-and-subscribe nature of my widgets works much like the metaglue's notification system. I also use the idea of having many separate general-purpose widgets that all work together to provide an application with the specific knowledge it needs by picking and choosing the widgets that are needed for a particular task, or extending existing agents to meet a particular need.

2.2 Perceptual User Interfaces

Work in Perceptual User Interfaces has grown along side with advancements in Intelligent Environments. In fact there is a whole conference dedicated to the study of PUIs [6]. There are some relevant works from this field that have inspired my research.

Hiroshi Iishi's early work was instrumental in starting to think outside of the keyboard and mouse and push the notion of a user interface to include perception of the world as an interface. [22] He advocates three main changes to the standard GUI interaction paradigm:

1. Interactive Surfaces - Turning surfaces in the real world into an active interface in the world of the computer.

2. Coupling of Bits and Atoms - Having everyday objects represent digital information

3. Ambient Media - Using things like sound and light to convey information.

This work was very influential in starting work on PUIs and greatly influenced my own work as well.

The following are examples of specific, PUIs that have been implemented and that have been instrumental to the work presented in this thesis.

Tollmar and Persson's work on the "6^{th} Sense" project is a complete application that uses light to signify remote presence. [33] [34] The implemented and field tested version uses motion sensing to determine if a person is around a lamp. Its structure is very similar to my Perceptive Presence Lamp in that there are two lamps, connected through a network. If there is motion in one area, the other lamp will glow. This is very similar to the first half of my initial prototype for the Perceptive Presence Lamp. Tollmar and Persson further field tested the lamp concept and found it to be more well received than expected [34]. They attribute this to the warm feeling of the light and its non-intrusive nature.

The Look-To-Talk (LTT) system presented in [29] adds the concept of using gaze to interact with a computer system. In this paper, LTT was being used in the concept of an Intelligent Environment and the room occupant could look towards an iconic representation of the "agent" that he wanted to activate. In this Wizard of Oz study, users preferred the LTT interface of a system of pushing a button in order to talk to the agent or vocally addressing the agent.

The Digital Family Portraits project at Georgia Tech is another example of a PUI used to convey awareness information. [27] This application presented a digital picture frame that could illustrate information about an elderly relative such as if they ate, socialized, or slept in regular patterns over time. This information was conveyed as icons around the actual photo of the person being "monitored."

2.3 Communication Applications

My work has also been influenced by the quite popular real-time network communications applications currently being used. Many of these applications have notions of marking awareness state, and, as seen below, many users are taking advantage of

that state.

One of the earliest instant messaging applications was Zephyr on MIT's Project Athena. [14] Zephyr allows users of the Athena computing system to see which other users are currently logged in and their physical locations (as determined from computer names). Zephyr also allows for away messages (using the zaway command) that can be custom tailored for each user. For example, a user could be away from a workstation and leave a "message" for a friend that he is getting lunch at a favorite restaurant, while leaving a message for all other users that he is simply away and will be back shortly.

Currently the most popular instant messaging application is AOL's Instant Messanger (AIM). As of October 2002, AOL boasts over 100 million users of their system. [1] Current versions of AIM allow users to see how long other users have been idle at their workstations (away) and also see general "away messages" that a user posts about his current state. Several studies have been performed on the use of Instant Messaging. [28] [20]

In [28], Nardi et al discuss IM in the workplace, especially as used in a lightweight, asynchronous manner. They discuss interaction using IM including coordinating schedules and keeping in touch with friends, family, and coworkers. More interestingly though, they discuss the outeraction present in IM. This includes negotiating availability by being "signed on" to the messaging system. Users used this simple awareness to know when to stop by someone's office or when to call them. This illustrates another common feature of IM found in this study, the use of awareness information to help plan communications in other media (face to face or over the phone). Finally, they found the IM creates "Awareness Moments" when people feel connected to others just because they are listed as being online. Often seeing a name sign on would trigger a visual image of that particular person.

Grinter provides an analysis of IM from a completely different perspective – that of teenagers (ages 15-21). [20] She also finds a similar use of presence information among people of this age range. According to her study, teenagers often use IM to know the right time to contact a friend. The college-aged students in her study

extensively used IM away messages to tell others when they were available or away. Finally, she found that AIM cut down on "phone circles" because one knew when the people they were trying to reach were available.

2.4 Media Spaces

The work on Media Spaces in the 1990s took the idea of instant messaging beyond text and into the realm of sharing pictures and video among co-workers. In these systems, people could literally see others and what they were doing, and in most cases determine for themselves if that person was available.

The NYNEX Portholes system [17] provided still images of users on a website. If another user wanted to know if a particular individual was in, he could look at the webpage and see if the person was physically present of looked busy. The user could also look at a more detailed page for each user that contained a bar graph of activity calculated from image differences over time as well as contact information. According to their analysis, this system suffered from many privacy problems including "camera shyness, threat of surveillance, loss of control over privacy, and lack of control over video images." Many of these problems were due to the fact that images were being captured at random times, and users felt that this invaded their privacy.

The Ravenscroft Audio Video Environment (RAVE) at Xerox EuroParc is a prototypical example of a media space. [19] RAVE placed cameras and monitors at people's work places and allowed them five modes of interaction. First, a user could "sweep" all other locations, or a subset of locations. Sweep allowed a user to peer in on others workplaces for one second intervals. The second functions was "glance" and allowed users to get a three-second view of an workplace, similar to the effect of walking past an open door and glancing in. "Office Share" was a mode that allowed for an always on connection between two locations, much like the users were sharing an office. Finally, "vphone" allowed for a conversation to occur between two offices and usually followed a glance to see if a person was available. Again in this system, it was up to the user to determine if a person was available based on looking at a live

scene of a remote location.

Cruiser [7] was another video-based system, similar to RAVE implemented at Bellcore. However, Cruiser enforced a reciprocity rule. In the glance mode of this system, whenever a person was looking at someone else, the other person could see them. According to the paper, it became unclear when it was appropriate to perform a glance as the intrusion of opening a shared video connection seemed to high.

A recent entry into this area is that IBM BlueSpace project. [3] It creates an office space that has a large light hanging above it. This light is called the Status Light and is manually controlled by the user via a touch screen. It will turn red when they signal that they are busy and green when they are available. This application illustrates using color to convey presence information through lights. In user studies, it showed that people use the color information to know when not to bother someone, or when they are free. The system also provided icons showing a user's state based on sensors and active badges (sitting in front of the computer, standing, not present). It was found that the users preferred these automatic means of obtaining awareness information over time and chose not to manually change their lights as often during a four week user study.

2.5 Computer Vision Modules

Finally, my work builds upon research in computer vision. The interface level of this application is written to accept input from different tracking algorithms. It is currently interfaced to a 3D blob tracker as explained below.

2.5.1 Blob and Person Tracking

Over the past decade, many groups have been working on building reliable person trackers. With the great increase in computational power, algorithms are now able to run in real time and robustly track multiple people.

Early tracking systems used monocular cameras and used color to differentiate between people and background. [9, 36, 31, 21, 18] The approach used in the current

system, [12] utilizes stereo cameras which provide two views of a scene and can help to alleviate problems due to changing illumination (due to displays, weather, etc.).

The current approach integrated into the Perceptive Presence system (see [12] and [8] for more detail) estimates the background by comparing the current stereo images to a stereo model of the background. Then, color regions are extracted that correspond to the invariant chromatic component of human skin color. Faces and hands can be tracked from this data by finding skin colored ellipsoid blobs and following them over time using connected component analysis along with color histograms of the blob areas. Heads are identified based on the results of a face detector [35] run on the image data from the blob area.

2.5.2 Head Pose Tracking

Another important area of Person Tracking, once a person has been found, is head pose tracking. Philipe-Morency has created a six degree-of-freedom head pose tracker that can "automatically segment and track a user's head under large rotation and lumination variations." [26] This approach uses several base frames to estimate the current orientation of a head by observing the motion between frames and also comparing them to the base frames. Head tracking data is important in determining where a person is looking and if they are trying to interact with an object by looking at it.

Chapter 3

An XML Interface for Perceptive Presence

In order for a wide variety of widgets to use data from computer vision modules, there is an Interface Layer that allows widgets to subscribe to XML [2] messages that contain information about the objects the vision system can see or track. It is the job of the interface to:

1. Allow clients to subscribe to a particular set of data
2. Ensure that the appropriate vision modules are running based on the current set of user requests
3. Receive data from computer vision modules
4. Filter computer vision data for each client
5. Send filtered data in standardized XML packets to each client

Each of these functions will be discussed further in the Implementation section.

Data is represented as "blobs." A blob could be any number of different things, depending on the underlying computer vision application. For example, a blob could be a person, a face, a hand, a car, or, in the current implementation, flesh colored regions in the foreground. The particular data that is available to clients includes:

1. Unique blob ID – useful for tracking
2. Timestamp – when the blob was recorded
3. 3D Location – the current location of the center of the blob
4. 3D Velocity – the current velocity of the center of the blob
5. 3D size – the height, width, and depth of the blob
6. Face Information – a boolean value declaring if the blob is a face
7. 3D Orientation – the current angles of the object in 3D space describing the orientation of its "front"

Obviously, not all computer vision modules will implement all of these functions. The interface is designed to be generic, reusable, and representative of the needs of perceptive presence applications.

3.1 Implementation

The Interface Layer is implemented in C++ and compiled as a library so that any computer vision module should be able to use it with little integration effort. It uses the Xerces XML parser [38] which is freely available from Apache to parse and create XML messages.

3.1.1 Allow clients to subscribe to a particular set of data

In order to allow clients to subscribe to blob data, the interface layer opens a UDP socket on a known port (10900, in my implementation). The client sends a UDP datagram to this port containing "Setup Data" in XML. All setup data must conform to the setup schema as illustrated in Figure A.1. The message must include host and port fields specifying where the requested data will be sent to, and boolean values for the data that client wishes to receive. An example of a setup data message is shown below, in Figure 3-1.

```
<xml>
   <trackerSetup>
      <host>presence1.ai.mit.edu</host>
      <port>3136</port>
      <id>true</id>
      <timestamp>true</timestamp>
      <position>
         <x>true</x>
         <y>true</y>
         <z>false</z>
      </position>
      <velocity>
         <dx>true</dx>
         <dy>true</dy>
         <dz>false</dz>
      </velocity>
      <size>
         <width>true</width>
         <height>true</height>
         <depth>false</depth>
      </size>
      <headPose>
         <isAHead>true</isAHead>
         <phi>false</phi>
         <theta>false</theta>
         <angle>false</angle>
      </headPose>
   </trackerSetup>
</xml>
```

Figure 3-1: An example of the XML Setup Data send to the Interface Layer by a client interested in getting updates on X and Y position, velocity, and size as well as if the blob represents a head or not. This data will be sent periodically to presence1.ai.mit.edu on port 3136 whenever the vision system presents available data.

Upon receiving the packet, the Interface Layer parses the packet and stores the boolean values for each type of data according to the client's request in a data structure for that particular client.

3.1.2 Ensure that the appropriate vision modules are running based on the current set of user requests

Once a request has been received, the Interface level checks to see what services are necessary based on the data that is now in the client data structure. For example, if a user wanted the x and y blob positions, a blob tracker would have to be running or if a user wanted face angle, a head pose tracker would have to be running. The Interface indicates to the vision modules if they need to be started by returning a data structure describing the types of services currently requested. If the module can provide any of the newly requested services, it should start sending data as described below.

3.1.3 Receive data from Computer Vision Modules

There is a simple interface to the computer vision modules that makes integration an easy task. A method `newData(...)` is called each time there is new data available from the computer vision module. When a round of data is complete, the vision module should call `send()` indicating that all the data for that timestep has been added and can be packetized and sent to any interested subscribers.

3.1.4 Filter computer vision data for each client

Each time `send()` is called, the Interface Layer iterates through each of its clients. For each client, the Interface will iterate through each blob it has received since the last time `send()` was called. It will add the requested fields to XML messages containing all blobs in that round edited for each client's requested set of fields.

3.1.5 Send filtered data in standardized XML packets to each client

The Interface Layer sends a UDP Packet conforming to the specification in Figure A.2 to each client. The packet will contain only the information that the client is interested in so that bandwidth is not wasted. As these are UDP streams, it would be unfair to send a large quantity of large packets since many routers are not equipped with mechanism to allow UDP and TCP traffic to fairly compete for bandwidth. [11] A sample packet is shown below, in Figure 3-2.

3.2 Analysis

The Interface Layer seems to provide a good abstraction for representing data needed by Perceptive Presence applications. As shown in Chapter 4, a variety of Perceptive Presence Widgets can easily be created using the data provided by the interface. The interface is easily integrated into existing computer vision applications with only the addition of a constructor and two lines of code.

I created two different implementations of the XML parser. One created the XML messages by concatenating character arrays and the other used a freely available XML parser called Xerces [38]. Both implementations perform the same basic task, but the Xerces implementation is much slower due to the overhead required in creating node objects, verifying their contents, and parsing an XML tree into a character array. A summary of the performance of these two algorithms can be found in Figure 3-3.

Looking at the both sets of data, the running time doubles when the rate increases by a factor of two. However, it is very efficient to add more blobs to each packet, especially in the Xerces implementation. In that algorithm, going from 10 blobs/packet to 20 blobs/packet requires less than a 2% increase in processor usage. This is most likely due to the overhead in parsing a tree of nodes in the XML implementation. Therefore, if the application can afford to receive data less frequently, it is more efficient to put more than one time-block of data into a single packet. Unfortunately this

```
<xml>
  <blobs>
    <blob>
      <id>21</id>
      <timestamp>1035568444265</timestamp>
      <location>
        <x>39</x>
        <y>50</y>
      </location>
      <velocity>
        <dx>5</dx>
        <dy>3</dy>
      </velocity>
      <size>
        <width>11</width>
        <height>14</height>
      </size>
      <headPose>
        <isAHead>true</isAHead>
      </location>
    </blob>
  </blobs>
    <blob>
      <id>22</id>
      <timestamp>1035568444265</timestamp>
      <location>
        <x>50</x>
        <y>10</y>
      </location>
      <velocity>
        <dx>8</dx>
        <dy>−5</dy>
      </velocity>
      <size>
        <width>5</width>
        <height>6</height>
      </size>
      <headPose>
        <isAHead>false</isAHead>
      </location>
    </blob>
  </blobs>
</xml>
```

Figure 3-2: An example of two blobs with information corresponding to the query example in Figure 3-1.

overhead results in very poor performance in the Xerces implementation. Sending 5 blobs per packet at 5 frames per second requires 16.5% of the CPU and sending at 10 frames per second requires roughly double that at 33.4%.

However, with the character array implementation, it is much more efficient to send XML data, and quite practical to include this interface in even very computationally demanding computer vision algorithms. Sending 10 blobs per frame at 5 frames per second only takes 0.15% CPU and sending those same 10 blobs at 10 frames per second takes exactly double that at 0.30% CPU.

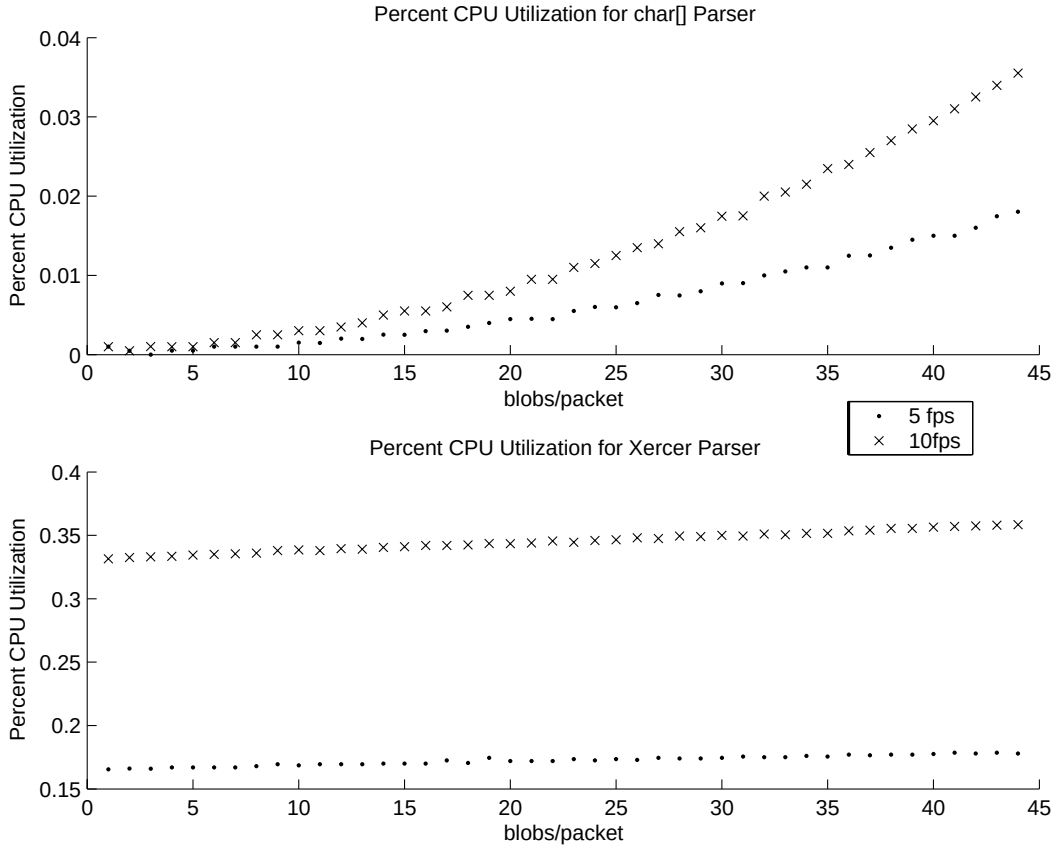

Figure 3-3: A plot showing percent processor utilization for various numbers of blobs per packet for both 5fps and 10fps operation (Currently runs about about 6-7 fps with the blob tracker). Data was calculated by measuring the time to send 5 (or 10) rounds of n blobs per second for thirty seconds and averaged.

Chapter 4

Perceptive Presence Widgets

Widgets are the heart of the Perceptive Presence system; they turn blobs into useful contextual concepts that an application can understand and work with. For example, a Widget could say if a person is present or not, or even where they are located and if how they are moving seems to correspond to a given activity or action. It could say that there is a person looking at the camera, or looking a a point two feet left of the camera. It could even say that a person was "selecting" a virtual object by moving into a given area. All these things are possible when the data from the interface is further abstracted.

4.1 Definition

By definition, Widgets sit between the Interface Layer and the Application Layer. Their job is to take blob data from the Interface and translate it into information about the people and activities that are within the view of the camera, or within a specific region of interest, called an Area. There are two main types of widgets: vision-centric widgets and application-centric widgets. The vision-centric widgets work to improve the accuracy of the data that comes from the underlying vision system or to infer contextual state (for example, by taking head tracking data and analyzing the motion to see if a user is nodding his head). In contrast, the application-centric widgets sit on top of the vision-centric widgets and create useful abstractions for

application developers (for example a spatial selection tool with an interface much like a GUI list that changes its selection when people enter and leave physical Areas).

4.2 Implementation

Widgets can be wirtten in any language that can accept datagrams such as C++, C#, or Java. The Widgets that were built for this thesis are implemented as Java classes that extend a common abstract Widget class. The abstract class provides for common functionality, such as connecting with the interface layer and parsing the received XML packets into a more usable data structure. There is also a mechanism for widgets to inform other widgets or applications about each other. Widgets or applications can subscribe to other Widgets and receive information each time their state changes. This state update occurs via an XML text message sent from the widget to all of its listeners.

For example, let's say there are two widgets, a Person Present Widget and a Spatial Selection Widget and the Person Present Widget will tell you if someone is in an area at a given instant. The Spatial Selection Widget can receive updates from several Person Present Widgets all listening to different areas to determine if there are people in any of the areas of interest. The Spatial Selection Widget would then determine which "items" are selected based on where people are located and pass that information back up to the application anytime the "selection" changes.

4.2.1 Areas

Widgets operate over regions of interest called Areas. Areas are three dimensional boxes in space and a Widget will only process blobs that appear within this space. This allows for Widgets to be created to watch over activities only in certain parts of the scene, such as in front of a desk, without worrying about blobs in other parts of the scene. This is a powerful tool in order to understand what is happening in different zones of a room. Areas can be defined by hand, or created during run time with a tool such as the Activity Map creator in [12].

4.3 Types of Widgets

There a many types of Widgets that are possible to create using this framework. I will discuss four of them here. The first is called the Person Present Widget and will give instantaneous information about a person's presence in an Area of interest. The second is the Person Around Widget and subscribes to the Person Present Widget. The Person Around Widget will give information about a person's assumed state and if they are likely around and will soon return to the area of interest if they are not instantaneously present. I will also introduce a Spatial Selection Widget that allows a user/users to select between physical areas based on their physical 3D position(s) and a head Gaze Selection Widget that performs a similar task based on position and head angle.

4.3.1 Person Present

The goal of the Person Present Widget is to determine if there is a person present at the moment in the given Area of interest. The widget should send an update each time a person enters or leaves the specified Area.

Implementation

The Person Present Widget runs an Exponential Weighted Moving Average (EWMA) over the number of blobs present in each frame, as received from the blob tracker throught the XML Interface, according to Equation 4.1

$$average = 0.9 * average + 0.1 * numBlobs \tag{4.1}$$

This update at each frame allows for quick convergence to the true value of number of blobs present in a given frame without responding too quickly to frames that show no blobs or have an occasional false detection. When the average is above a threshold α, then it is declared that a person is present. When the average drops below β it is declared that the person has left. In the current implementation, $\alpha = 2$ and $\beta = 1$.

The results of this method can be seen in the Analysis section.

To improve the performance of the above algorithm for determining the average, it was noted that the average takes over 15 frames to fall below 1 when starting from a typical "present" value of 5. That is almost three seconds at 5fps. To correct this deficiency, it is noted that when a person is leaving a scene, he is last seen near the edge of the scene and moving out of the scene. This information is available to us already through the interface. In the "improved" Person Present Widget, if there are no blobs present at a given time and in the last frame, the average blob position was within 10% of the edge and moving with a velocity out of the scene, the average updates according to Equation 4.2; otherwise, it updates according to Equation 4.1 just as before. This leads to much faster convergence when people leave an Area as seen below.

$$average = min(0.9 * average, 1) \tag{4.2}$$

Analysis

In order to analyze the effectiveness of this widget, experiments were performed where seven people of varying age, color, and sex performed tasks at a work area. The range of the camera was split into two regions. Area A covered the left half of the field of view and went approximately 2 meters deep. Area B encompassed the rest of the field of view. A Person Present Widget was run on each area. The participants were asked to enter the work area, sit down in front of a computer (Area A), read a news story of their choice from CNN.com, move away from their desk (into Area B), return to the desk, and then leave. Data was collected about their average position, average velocity, number of flesh colored blobs detected, and whether the Widget thought that they were in Area A or B.

Figures 4-1 and 4-2 show an example of the results for Area A for a given participant using Equation 4.1 for the update rule. Figure 4-1 illustrates the common behavior when a person enters or leaves and Figure 4-1 shows the average number of blobs and the current present state as determined by the Widget (In all figures

"Present" is the high state). As you can see, this algorithm converges quite quickly when a person enters (declaring that they are present in about 4 frames - less than one second); however, it is quite slow at stating that they have left due to the fact that we want to be cautious about saying someone is not present when they really are or might have their face obstructed for a second or two. For this case, it took 23 frames (greater than four seconds) to declare that someone has left.

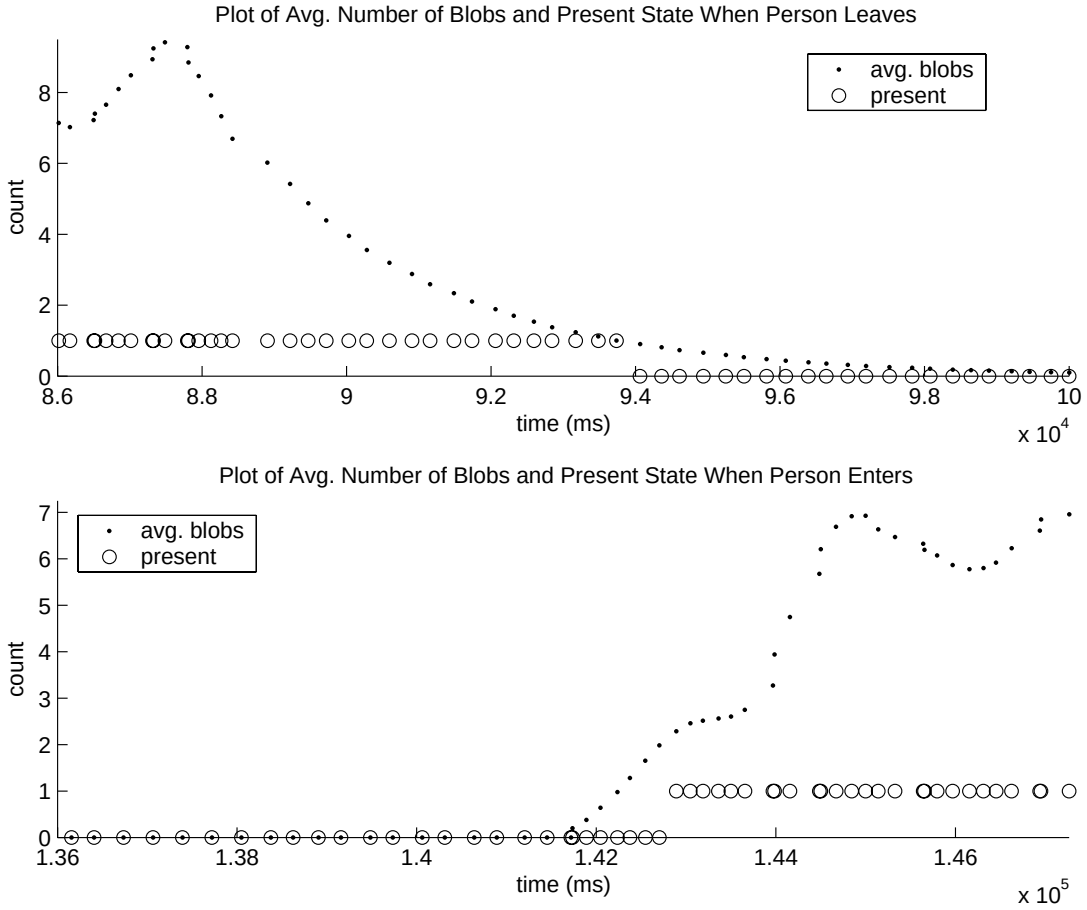

Figure 4-1: This graph illustrates the behavior of the simple Person Present Widget using only Equation 4.1 to update the average each frame. It is easy to see when the person entered and left the area. (Note: A person is declared to be present when the presence level is "high")

Figures 4-3 and 4-4 show an example of the results for Area A for a given participant using Equation 4.2 for the update rule. As you can see, this algorithm converges quite quickly in both directions, declaring that someone has left in just two frames (less than half a second). Clearly, using the position and velocity data helped in

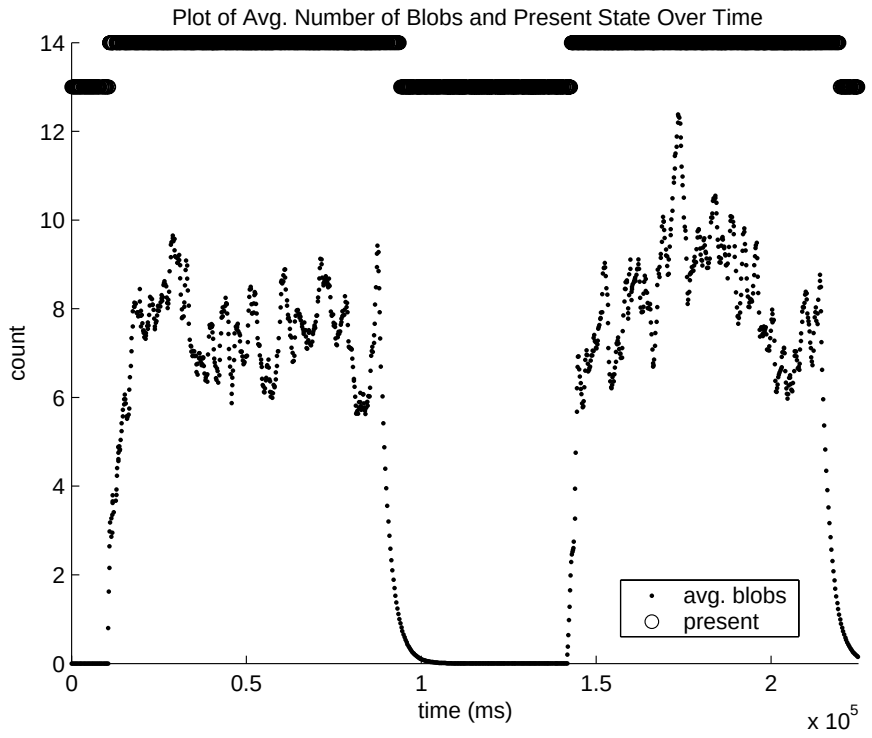

Figure 4-2: This graphs shows the average number of blobs used to declare if a person was present. (Note: A person is declared to be present when the presence level is "high")

understanding the dynamics of the scene.

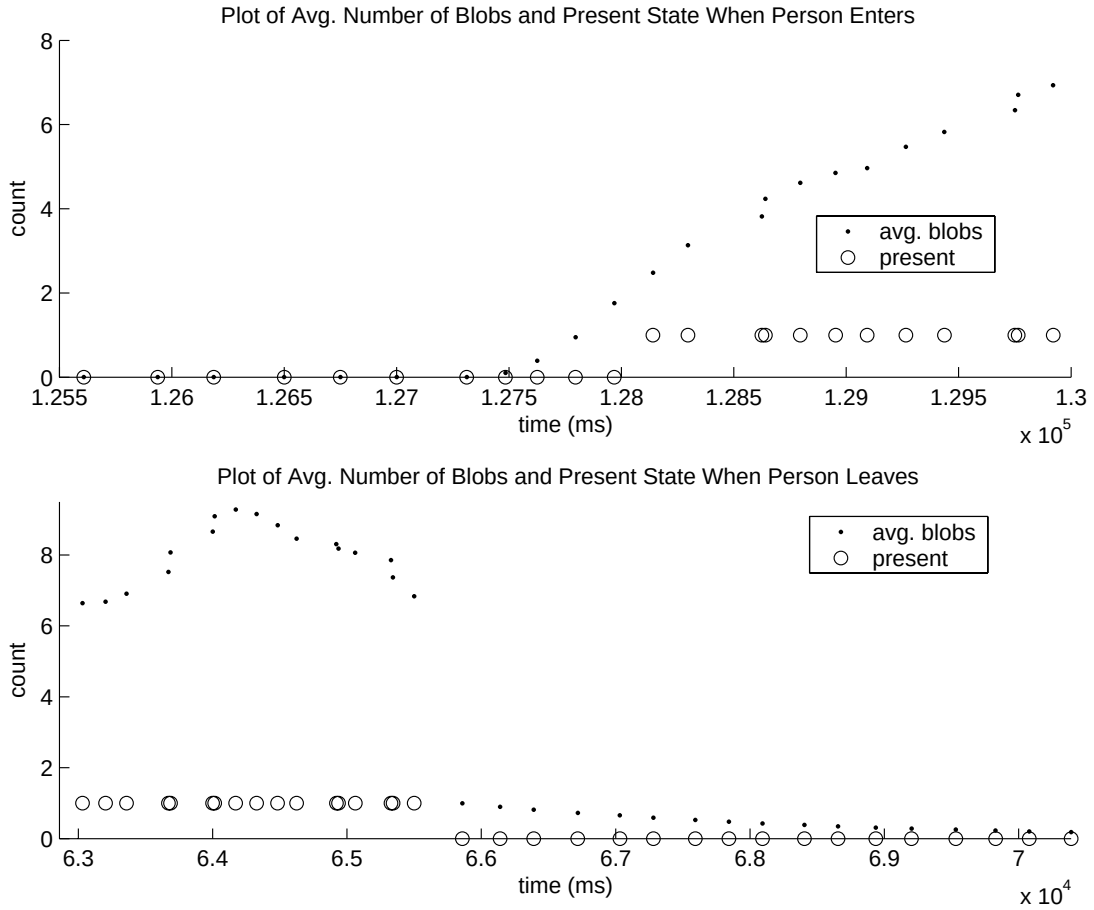

Figure 4-3: This graph illustrates the behavior of the enhanced Person Present Widget using position and velocity data and Equation 4.2 along with Equation 4.1 to update the average each frame. It is easy to see when the person entered and left the area. (Note: A person is declared to be present when the presence level is "high")

The average results from these experiments can be seen in Table 4.1. Note that with the use of position and velocity information, the Widget can declare quite quickly (mean 0.61 seconds, median 0.4 seconds) if someone has left.

4.3.2 Person Around

The goal of the Person Around Widget is to determine if a person is likely "around" at a given time. Around is a subjective term roughly meaning "probably nearby" to a given area of interest, A, and includes times when a person just steps away from

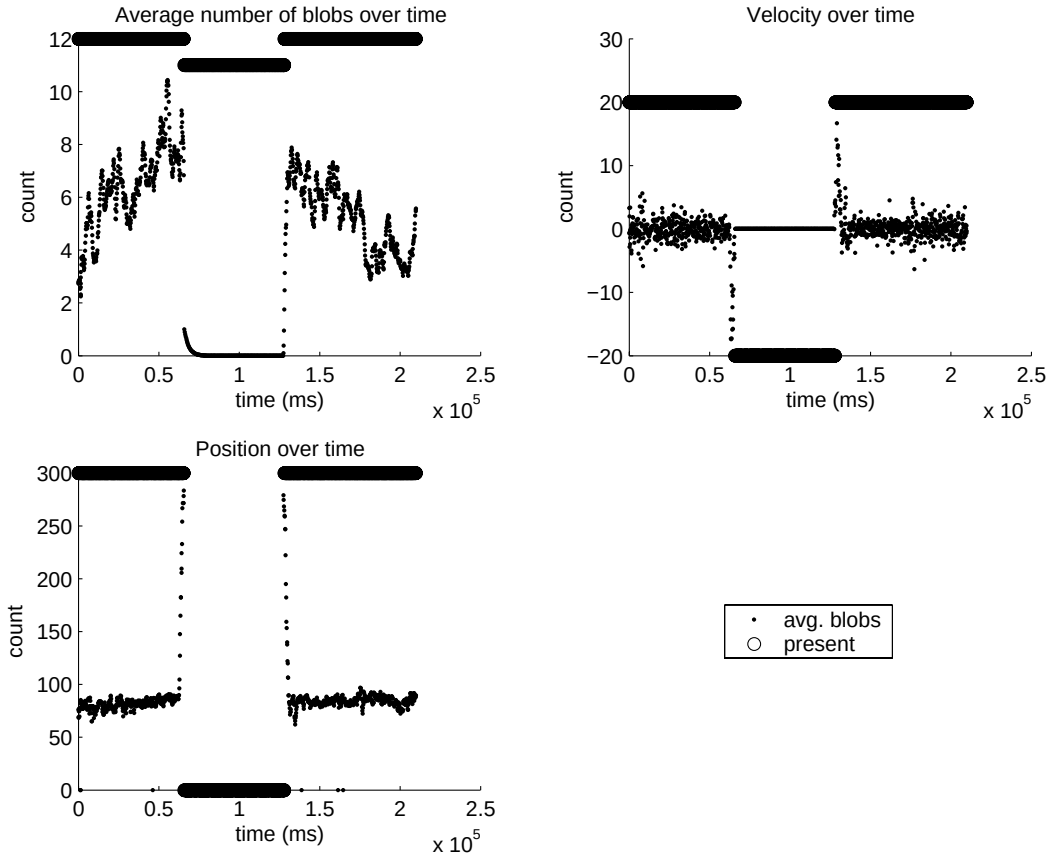

Figure 4-4: These graphs show the raw data used to declare if a person was present. This includes (from top left) average number of blobs, average velocity, and average position of the blobs. (Note: A person is declared to be present when the presence level is "high")

Task	**Mean Time** (sec)	**Median Time** (sec)	**Std. Error**
Entering	1.43	0.8	0.40
Leaving	0.61	0.4	0.15

Table 4.1: Average Results of the Person Present Widget (over 35 trials for 7 users of varying age, color, and sex)

their desk for a minute or two or goes to another location in the room other than A.

Implementation

The Person Around Widget is implemented by extending the Person Present Widget. Whenever the Person Present Widget finds a person present in the desired area of interest, it passes a message to the Person Around Widget which then declares that someone is present. If a person leaves that area, the Person Around Widget waits one minute before declaring that they have left. If they return anytime in that minute and leave again, they are given another whole minute to return before they are declared away.

A second version of this Widget increases this away timer by one minute each time the person goes away and then returns within five minutes. The timer will increase to a maximum of five minutes. Whenever a person stays for more than five minutes, the timer returns to one minute the next time they leave. It is assumed that if a person is coming and going frequently, their true state is "present."

Analysis

The times chosen for the Widget are not arbitrary and are a result of user tests performed while developing this Widget. A camera was set up in two offices for a day each and the participants were asked to keep a log of when they considered themselves to be in and when they considered themselves to be away. The Person Present Widget was run throughout the day to find when people were physically present at their desks. It was seen that there are often one to two minute breaks where a person considers himself to be "in" when he really is away from his desk and that when someone is in and out several times in a row within a few minutes of each other, they consider themselves to be "in." A final observation was that there is often a few second gap, when the Person Present Widget thinks no one is present that is caused by a person turning around, blocking their face, or otherwise not visible by the camera. The same smoothing that helps when people come and go quickly also helps in these cases, drastically improving reliability.

Once the parameters were set, the final Person Around Widget was deployed into offices within the vision group at the AI Lab and tested for a full day in each location. Participants again kept a log of when they considered themselves to be in or out and the performance of the Widget was compared with that log.

The two offices used were quite different in order to get a sense of how this Widget might work in different environments. One setting, O1, was in a common space at a desk used solely by the participant, but with other people working within the range of the camera and people walking past the camera throughout the day. The second office, O2, was a two-person shared office where the non-participant was always within the field of view of the camera.

The Widget performed quite well in both situations by using Areas that were limited in depth to only catch a person sitting at the desk, close to the camera, when determining if that person was present. The results of the two-day study can be seen in Table 4.2. The error rate using this approach on average was 53% of the error rate using just the Person Present Widget.

Office	**Accuracy with Person Present**	**Accuracy with Person Around**
O1	92.5%	95.4%
O2	90.6%	95.9%
Avg	91.7%	95.6%

Table 4.2: Results from a study of the Person Around Widget. The accuracy is determined by the percentage of seconds the system was incorrect in determining if a person was present at their desk. Average accuracy is calculated to be the average over all time for all participants.

Common errors included declaring someone had left for a minute or two while a person was on the phone or otherwise looking away from the camera (no flesh color in view), thinking someone would return and waiting all five minutes before declaring they had left, and sometimes not noticing when a visitor was present due to their angle (no flesh present) or position in the room. All of these errors were rare though as can be seen from the total accuracy of the system at 95.6 percent. Because the results were so similar across different people with different offices and schedules, no further tests in this widget were performed. Tests of the Perceptive Presence Lamp

in Chapter 5 continue to validate the success of this Widget.

4.3.3 Spatial Selection Widget

The Spatial Selection Widget is a general interface component that can be used to implement spatial selection tasks. Given a set of named areas, this widget will inform its listeners which areas currently have people in them. This widget could be used by application developers in a similar manner to the current GUI list widget that allows for single and multiple selection.

Implementation

The Spatial Selection Widget has a rather simple implementation. The user provides a number of named Areas and the widget creates a Person Present Widget for each area. When a person enters or leaves a given area, the widget will send an XML message to its listeners that contains the current state of all Areas of interest. An example of such a message for areas labeled "desk," "door," and "couch" is shown in Figure 4-5.

```
<xml>
   <SpatialSelection>
      <desk>true</desk>
      <door>false</door>
      <couch>true</couch>
   </SpatialSelection>
</xml>
```

Figure 4-5: Example data sent to listeners of a Spatial Selection Widget for a room where there is a person at their desk and someone on the couch. The areas were labeled "desk," "door," and "couch" when initializing the widget.

Analysis

A simple test application was built that used this widget to select between squares on a screen (as seen in Figure 4-6). Four Areas were defined. These were:

1. The left half of the camera's field of view with a depth from 0-3 feet.

2. The right half of the camera's field of view with a depth from 0-3 feet.

3. The left half of the camera's field of view with a depth from 3 feet to infinity.

4. The right half of the camera's field of view with a depth from 3 feet to infinity.

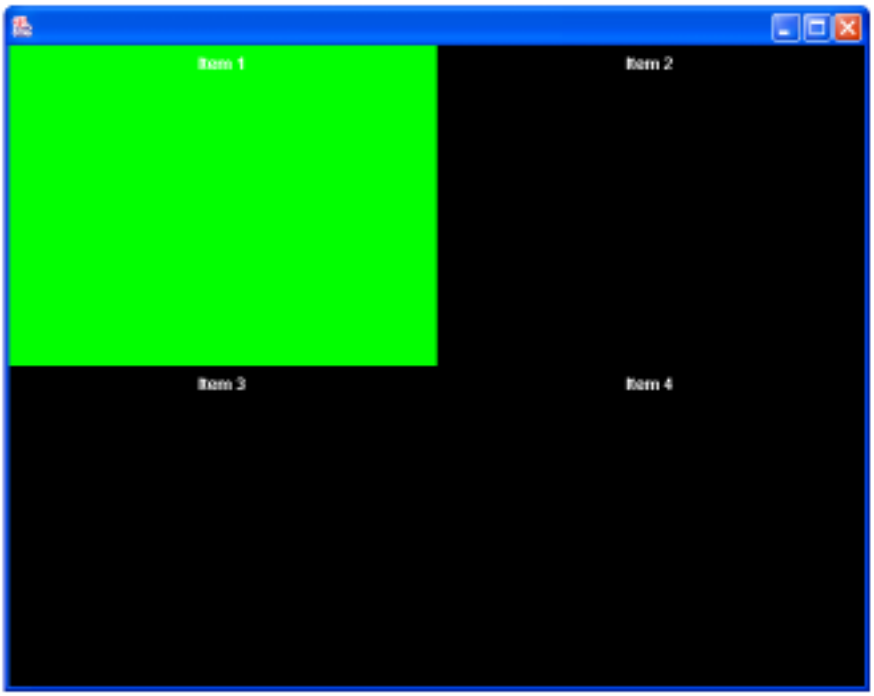

Figure 4-6: A simple example application using the Spatial Selection Widget to choose among four items. The four squares respond to the right and left foreground and background of the camera's field of view. Currently there is a person in Area 1, the left foreground.

This application demonstrated that the effort to integrating a perceptive presence widget into an application is little more than that required to add a GUI component and was also a visual way to demonstrate the power of the widget set.

This widget has the same response times as the Person Present Widget when people enter and leave an Area.

4.3.4 Gaze Selection Widget

The Gaze Selection Widget is another example of an application-centric widget. It is similar to the Spatial Selection Widget except it uses head pose to select based on

where a person is looking. This can be used for applications that require navigating via head gestures such as looking at objects to activate them (i.e. a light switch or a heating control) or navigating computer interfaces by looking in certain directions.

Implementation

A Gaze Selection Widget listens to data about head pose through the Vision Interface. Just like the Spatial Selection Widget, Areas are provided by the user. Every time the widget receives new blob information for a blob that is identifued from the XML Interface to be a head, it calculates the vector from the blob in the direction of the head pose. It then determines if this vector enters any of the areas of interest and returns an XML message to all of its listeners like the one shown in Figure 4-7 if any high level selection state has changed.

```
<xml>
   <GazeSelection>
      <lightSwitch>false</lightSwitch>
      <door>false</door>
      <window>true</window>
   </GazeSelection>
</xml>
```

Figure 4-7: Example data sent to listeners of a Gaze Selection Widget for a room where there is one person seated in front of the camera looking at the window, an area defined when the widget was constructed. The areas are labeled "lightSwitch," "door," and "window."

Chapter 5

Presence Lamp: A Perceptive Presence Application

The Perceptive Presence Lamp was introduced in Chapter 1. It is a lamp and camera that aim to virtually connect two places by conveying awareness information about each location. This chapter will discuss a version of the lamp that uses the Perceptive Presence Widgets in order to obtains its awareness information.

Figure 5-1: The color changing Perceptive Presence Lamp uses a stereo camera and the Spatial Selection Widget to determine where people are located and conveys this information to a remote location through of color on a lamp.

This lamp, as shown in Figures 5-1 and 5-2, changes color depending on the state of the person/people at the remote location. If the lamp is off, there is no one present.

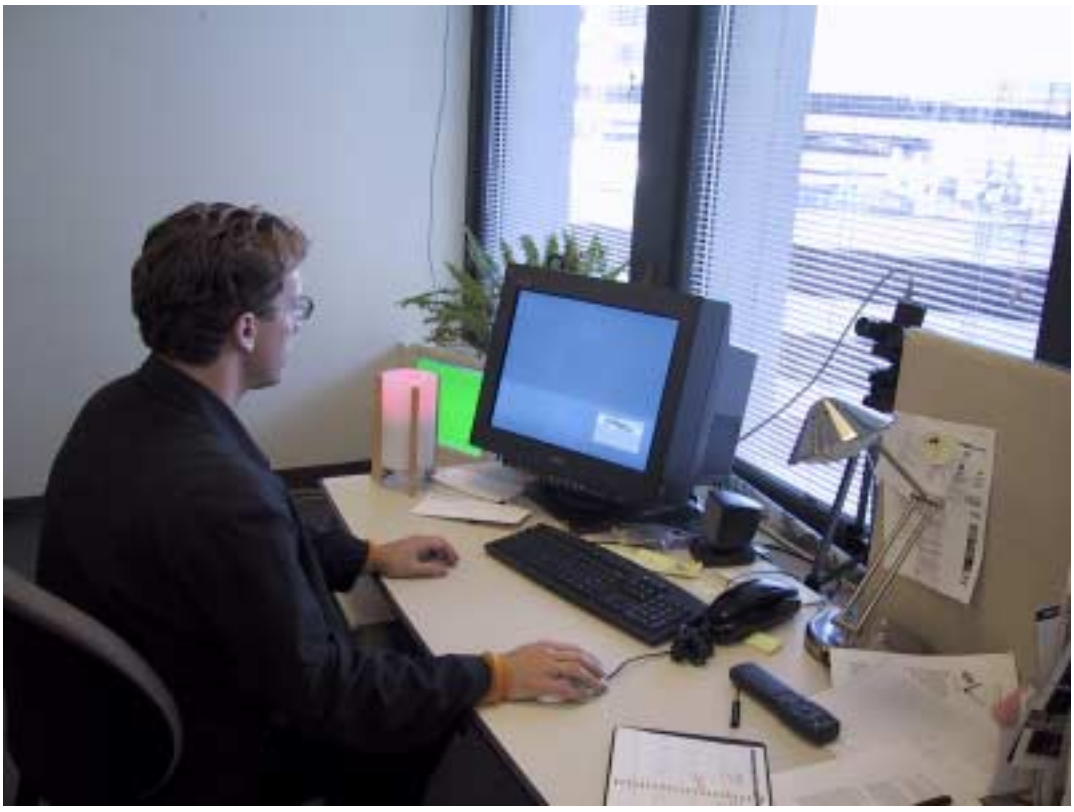

Figure 5-2: A user is sitting at his desk (his local presence display monitor in the back is green) and the person he is remotely connected to is currently busy (the lamp is red).

State	Color Changing Lamp Action
A person is present at their desk	green
There are multiple people present – the person is busy	red
The person is interacting with his lamp – he is trying to get the user's attention	blue
No one is present	off

Table 5.1: States of the Widget-Based Perceptive Presence Lamp

If it is green, someone is present and working at their desk. When guests are present, the lamp turns red to signify that the user is busy and finally when the user moves in front of his lamp, the other lamp turns blue signifying that the user wishes to get the remote user's attention. These states are summarized in Table 5.1.

The color changing in this system is very similar to the Status Lights in the BlueSpace/myTeam system [3] which turn green when a user sets his state to available and red when they set their state to busy; however, with this system, all awareness information is automatically computed and no user interaction is required.

5.1 Implementation

The Perceptive Presence Lamp is implemented using the Spatial Selection widget. Three areas are defined: One area is the "desk" region and consists of the space directly in front of a person's desk where they would be sitting while they are working. The second area is the "visitor" area. The visitor area consists of all the other space in the office that is viewable by the camera. The final area is the "interacting" area and consists of the space directly in front of the camera/lamp and is the place the user goes to interact with the lamp to get the remote user's attention. The lamp application utilizes the Person Present and Person Around Widgets to determine if a person is instantaneously present in the visitor or interacting regions and if someone is likely around the desk region.

State	Message
A person is present at their desk	`<xml><update>present</update></xml>`
There are multiple people present – the person is busy	`<xml><update>busy</update></xml>`
The person is interacting with his lamp – he is trying to get the user's attention	`<xml><update>interact</update></xml>`
No one is present	`<xml><update>away</update></xml>`

Table 5.2: XML messages for updating Presence Lamp state

The lamp's vision processing software runs on a PC hidden under the desk and sends XML messages to the remote lamp's computer every time the state changes. The messages sent are summarized in Table 5.2. A state is assumed to exist until a new state is sent.

When a Lamp application receives a message, it updates the color of its lamp accordingly. The lamps are Color Kinetics color-changing LED lights placed into the body of a lamp from IKEA and controlled through a Color Kinetics "Smart Jack" USB interface.

5.2 Evaluation

Since the Perceptive Presence Lamp uses the Spatial Selection Widget described above, its technical performance is identical to the performance of this widget, including the time it takes to identify that a person is present or has left.

We performed a series of user studies involving six people for one to two days each. Over 46 hours of data were collected from varying office conditions including shared work areas, shared offices, and other areas where only the user was visible in the field of view of the camera. For each space, a set of areas was manually defined to explicitly mark off the work area, visitor area, and interaction area in three dimensions. Users were asked to keep a log of what they considered their state to be throughout the day. This was compared with the lamp's choice of state. The results are summarized in Table 5.2 and show an average accuracy of over 97%.

Errors tended to be more false negatives than false positives. Some common errors

Office	Accuracy of Lamp	Length of Usage
O1	95.4%	7 hours
O2	95.9%	7 hours
O3	97.8%	16 hours
O4	99.0%	16 hours
Avg	97.6%	

Table 5.3: Results from a study of the Perceptive Presence Lamp Accuracy is the percentage of time the state of the lamp agreed with the state of the user.

included people considering themselves to be present when they were outside of their immediate work area (sitting on a couch or getting coffee) and the lamp stating that a person had left after they had their backs to the camera for over a minute (no flesh colored regions showing). The first error can be helped by having more complex areas or by learning areas through a system like the Activity Map learning presented in [12] and associating certain areas to "Present, "Away," "Busy," and "Interacting." The second error can be improved by having multiple cameras or a model other than flesh color for identifying people.

Overall, the users liked the concept of the lamp and used it to judge the availability of the other person. The lamp was frequently used to determine when a person returned from a meeting or lunch and saved wasted trips down the hall to keep checking to see if they had returned. Most importantly, it served as a successful test of an application using the Spatial Selection Widget.

5.3 Generalization of the Lamp

The concept of the Preceptive Presence Lamp can be generalized to other devices besides a lamp. Some of these concepts are shown in Figure 5-3.

The Perceptive Presence Display uses a projector to shine images onto a wall, table, or divider. The user can configure different images to correspond to different states, or even different users that might all be using the lamp. A small bar on the bottom of the display shows the current local state as viewed at the remote location(s).

Another version of the Lamp, the Perceptive Presence Beacon, uses the same

Figure 5-3: Alternative interfaces for the Perceptive Presence Lamp. (a) shows the Perceptive Presence Beacon – local and remote users present, (b) shows the Perceptive Presence Display – local user present, remote user away, and (c) shows the Perceptive Presence Messenger – local user (Frank) busy, remote user (Konrad) present.

color-changing light used in the lamp, but instead shines it onto a wall, a corner, or a workspace. This may create more of an effect of being involved with the person on the other side of the connection as the light will cover a larger area and not be confined to a physical device.

Finally, this concept can be applied to traditional, screen-based awareness applications to create an AIM-style [1] application, the Perceptive Presence Messenger, that automatically updates when people enter, leave, or are busy.

Chapter 6

Conclusion

This thesis has presented an architecture for Perceptive Presence applications. It has defined the layers of this architecture and has given examples of implementations at each of these layers as well as an example of a user tested Perceptive Presence application, the Perceptive Presence Lamp.

6.1 Future Work

The work presented in this thesis can be the basic architecture for many context-aware applications. The data gained from application-centric widgets can drive future PUI applications and can allow application designers to include perceptive functionality in similar ways to including GUI functionality today.

The concept of Areas, as presented in Chapter 4, is easily extendible to include learned activity zones as presented in [12]. These zones can be learned by processing data from a person tracker through the XML interface and an Activity Selection Widget can be created that works much like the Spatial Selection Widget to select based on activity. This would be more robust than the hand-crafted areas created for each location and could allow for more complicated area shapes.

Another possible extension would include creating more vision-centric widgets that operate on data from general-purpose vision modules such as blob trackers or head pose trackers to detect certain gestures such as head nods, waves, or more

complicated hand motions. These widgets could use more probabilistic models in their decision making processes and could then be tied to applications that check for these actions at certain times (for example, to confirm a dialog box, or continue with a text narration).

Finally, there is the opportunity for widgets to inform the vision system of certain actions taking place at the application level. This could be used to request particular vision modules. For example, if the application itself is computationally intensive and ran on the same machine as the vision software, the application could request vision modules that require less computation, or if an application wanted to run several widgets that could all utilize the same vision module, a more general vision module could be started instead of several, possibly computationally intensive specific modules.

6.2 Contributions

In this thesis I have:

- Created an architecture for Perceptive Presence systems.
- Enumerated a common language for sharing presence information from computer vision modules
- Developed a standards-based XML interface module that can interface easily to vision modules and presence widgets
- Defined and tested several widgets related to presence
- Developed the Perceptive Presence Lamp using this widget architecture and tested its performance and usefulness

I see this system as being useful for many future applications including usage as the main vision interface for current context-aware spaces such as the Intelligent Room project. The main benefits to using this architecture are ease of development for application writers (abstracting away computer vision libraries and functions into

simple XML-based events), and the ability to reuse application level code as implementations of vision modules change.

Appendix A

XML Schema

A.1 Schema for Requesting a Stream of Blob Data

The XML Schema for sending a request to the server is seen in Figure A.1. The host and port fields are the hostname (in hostname format or dotted IP notation) and port that will be receiving the data. All other fields are boolean values which will determine what data is sent to that interface.

A.2 Schema for Representing Blob Data

The XML Schema for sending a request to the server is seen in Figure A.2. A given XML message may have more than one blob in it, and it will only contain the fields that were set to true on the initial Setup request for that particular host/port pair.

```
<?xml version="1.0"?>
<schema>
  <complexType name="trackerSetup">
    <sequence>
      <element name="host" type="string"/>
      <element name="port" type="integer"/>
      <element name="id" type="boolean"/>
      <element name="timeStamp" type="boolean"/>
      <element name="location" type="location"/>
      <element name="size" type="size"/>
      <element name="velocity" type="velocity"/>
      <element name="headPose" type="headPose"/>
    </sequence>
  </complexType>

  <complexType name="location">
    <sequence>
      <element name="x" type="boolean"/>
      <element name="y" type="boolean"/>
      <element name="z" type="boolean"/>
    </sequence>
  </complexType>

  <complexType name="velocity">
    <sequence>
      <element name="x" type="boolean"/>
      <element name="y" type="boolean"/>
      <element name="z" type="boolean"/>
    </sequence>
  </complexType>

  <complexType name="size">
    <sequence>
      <element name="width" type="boolean"/>
      <element name="height" type="boolean"/>
      <element name="depth" type="boolean"/>
    </sequence>
  </complexType>

  <complexType name="headPose">
    <sequence>
      <element name="isAHead" type="boolean"/>
      <element name="phi" type="boolean"/>
      <element name="theta" type="boolean"/>
      <element name="angle" type="boolean"/>
    </sequence>
  </complexType>
</schema>
```

Figure A-1: Schema for requesting a stream of blob data

```
<?xml version="1.0"?>
<schema>

  <complexType name="blobs">
    <sequence>
      <element name="blob" type="blob" minOccurs="0" maxOccurs="*"/>
    </sequence>
  </complexType>

  <complexType name="blob">
    <sequence>
      <element name="id" type="string" minOccurs="0" maxOccurs="1"/>
      <element name="timeStamp" type="long" minOccurs="0" maxOccurs="1"/>
      <element name="location" type="location" minOccurs="0" maxOccurs="1"/>
      <element name="size" type="size" minOccurs="0" maxOccurs="1"/>
      <element name="velocity" type="velocity" minOccurs="0" maxOccurs="1"/>
      <element name="headPose" type="headPose" minOccurs="0" maxOccurs="1"/>
    </sequence>
  </complexType>

  <complexType name="location">
    <sequence>
      <element name="x" type="integer" minOccurs="0" maxOccurs="1"/>
      <element name="y" type="integer" minOccurs="0" maxOccurs="1"/>
      <element name="z" type="integer" minOccurs="0" maxOccurs="1"/>
    </sequence>
  </complexType>

  <complexType name="velocity">
    <sequence>
      <element name="x" type="integer" minOccurs="0" maxOccurs="1"/>
      <element name="y" type="integer" minOccurs="0" maxOccurs="1"/>
      <element name="z" type="integer" minOccurs="0" maxOccurs="1"/>
    </sequence>
  </complexType>

  <complexType name="size">
    <sequence>
      <element name="width" type="integer" minOccurs="0" maxOccurs="1"/>
      <element name="height" type="integer" minOccurs="0" maxOccurs="1"/>
      <element name="depth" type="integer" minOccurs="0" maxOccurs="1"/>
    </sequence>
  </complexType>

  <complexType name="headPose">
    <sequence>
      <element name="isAHead" type="boolean" minOccurs="0" maxOccurs="1"/>
      <element name="phi" type="float" minOccurs="0" maxOccurs="1"/>
      <element name="theta" type="float" minOccurs="0" maxOccurs="1"/>
      <element name="angle" type="float" minOccurs="0" maxOccurs="1"/>
    </sequence>
  </complexType>
</schema>
```

Figure A-2: Schema for representing blob data

Bibliography

[1] AOL Instant Messanger, http://www.aim.com/.

[2] Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, and Eve Maler. Extensible markup language (xml) 1.0 (second edition). Technical report, W3C, October 2000.

[3] P. Chou, M. Gruteser, J. Lai, A. Levas, S. McFaddin, C. Pinhanez, and M. Viveros. Bluespace: Creating a personalized and context-aware workspace. Technical Report RC 22281, IBM Research, 2001.

[4] Michael Coen, Brenton Phillips, Nimrod Warshawsky, Luke Weisman, Stephen Peters, and Peter Finin. Meeting the computational needs of intelligent environments: The metaglue system. In *Proceedings of MANSE'99*, Dublin, Ireland, 1999.

[5] Philip R Cohen, Adam J Cheyer, Michelle Wang, and Soon Choel Baeg. An open agent architecture. In *AAI Spring Symposium*, pages 1–8, 1994.

[6] Conference on Perceptual User Interfaces, http://www.pui.org/.

[7] C. Cool, R.S. Fish, R.E. Kraut, and C.M. Lowery. Iterative deisgn of video communication systems. In *Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work*, pages 25–32, 1992.

[8] T. Darrell, D. Demirdjian, N. Checka, and P. Felzenszwalb. Plan-view trajectory estimation with dense stereo background models. In *2001 International Conference on Computer Vision*, 2001.

[9] T. Darrell, P. Maes, B. Blumberg, and A. Pentland. A novel environment for situated vision and behavior, 1994.

[10] Trevor Darrell, Konrad Tollmar, Frank Bentley, Neal Checka, Louis-Phillipe Morency, Ali Rahimi, and Alice Oh. Face-responsive interfaces: From direct manipulation to perceptive presence. In G. Borriello and L. E. Holmquist, editors, *Proceedings of UbiComp 2002: Ubiquitous Computing*, pages 135–151, Göteborg, Sweden, September 2002. Springer-Verlag.

[11] A. Demers, S. Keshav, , and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Internetworking: Research and Experience*, 1(1):3–26, 1990.

[12] David Demirdjian, Konrad Tollmar, Kimberle Koile, Neal Checka, and Trevor Darrell. Activity maps for location-aware computing. In *IEEE Workshop on Applications of Computer Vision, Orlando, Florida*, 2002.

[13] M Dertouzos. The oxygen project. *Scientific American, vol 282, no 3, pp. 52-63*, 1999.

[14] C. Anthony DeUafera, Mark W. Eichin, Robert S. French, David C. Jedlinsky, John T Kohl, and William E. Sommerfeld. The zephyr notification service. In *Proceedings on USENIX 1998*, 1988.

[15] A.K. Dey, D. Salber, and G.D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction (HCI) Journal*, 16(2–4):97–166, 2001.

[16] Anind K. Dey. Providing architectural support for building context-aware applications. Phd thesis, Georgia Institute of Technology, November 2000.

[17] Paul Dourish and Sara Bly. Portholes: supporting awareness in a distributed work group. In *Conference Proceedings on Human Factors in Computing Systems*, pages 541–547, Monterey, California, May 1992.

[18] A. Elgammal, D. Harwood, and L. S. Davis. Nonparametric background model for background subtraction. In *6th European Conference of Computer Vision*, 2000.

[19] William Gaver, Thomas Moran, Allan MacLean, Lennart Lovstrand, Paul Dourish, Kathleen Carter, and William Buxton. Realizing a video environment: Europarc's rave system. In *Proceedings of the Conference on Computer-Human Interaction (CHI-92)*, May 1992.

[20] Rebecca E. Grinter. Hanging out with computers: The role of im in teenage communication. In *Human Computer Interaction Consortium*, 2001.

[21] I. Haritaoglu, D. Harwood, and L.S. Davis. W4: Real-time surveillance of people and their activities. *PAMI*, 22(8):809–830, August 2000.

[22] H. Iishi and B. Ullmer. Tangible bits: Towards seamless interfaces between people, bits, and atoms. In *Proceedings of ACM CHI'97 Conference on Human Factors In Computing Systems*, pages 234–241, 1997.

[23] M Lombard and T.B. Ditton. At the heart of it all: The concept of presence. *At the heart of it all: The concept of presence*, 1997.

[24] Matthew Lombard, Theresa B. Ditton, Daliza Crane, Bill Davis, Gisela Gil-Egui, Karl Horvath, and Jessica Rossman. Measuring presence. *Presence 2000: The Third International Workshop on Presence*, 2000.

[25] Marvin Minsky. Telepresence. *Omni*, June 1980.

[26] Louis-Philippe Morency, Ali Rahimi, Neal Checka, and Trevor Darrell. Fast stereo-based head tracking for interactive environment. In *Proceedings of the Int. Conference on Automatic Face and Gesture Recognition*, 2002.

[27] Elizabeth D. Mynatt, Jim Rowan, Annie Jacobs, and Sarah Craighill. Digital family portraits: Supporting peace of mind for extended family members. In *Procedings of CHI'01*, 2001.

[28] Bonnie A. Nardi, Steve Whittaker, and Erin Bradner. Interaction and outeraction: instant messaging in action. In *Computer Supported Cooperative Work*, pages 79–88, 2000.

[29] Alice Oh, Harold Fox, Max VanKleek, Aaron Adler, Krzysztof Gajos, Louis-Philippe Morency, and Trevor Darrell. Evaluating look-to-talk: A gaze-aware interface in a collaborative environment. In *Proceedings of the Conference on Human Factors in Computing System (CHI)*, 2002.

[30] Brenton Phillips. Metaglue: A programming language for multi-agent systems. Meng thesis, Massachusetts Institute of Technology, 1999.

[31] C. Stauffer and W.E.L. Grimson. Learning patterns of activity using real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8), 2000.

[32] Charles Steinfield, Chyng-Yang Jang, and Ben Pfaff. Supporting virtual team collaboration: The teamscope system. In *Procedings of GROUP'99: International ACM SIGGROUP Conference on Supporting Group Work*, 1999.

[33] Konrad Tollmar, Stefan Junestrand, and Olle Torgny. Virtually living together - using multiple-method design in the search for telematic emotional communication. In *Proceedings of DIS 2000*, 2000.

[34] Konrad Tollmar and Joakim Persson. Understanding remote presence. In *Procedings of NordiCHI'02*, 2002.

[35] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001.

[36] C.R. Wren, A. Azarbayejani, T.J. Darrell, and A.P. Pentland. Pfinder: Real-time tracking of the human body. *PAMI*, 19(7):780–785, July 1997.

[37] X10, http://www.x10.com/.

[38] Xerces XML Parser, http://xml.apache.org/xerces-c/index.html.