HITSI: A Human Interaction through Sensor Interpretation Application

Department of Computer Science and Computer Engineering

Latrobe University Bundoora
Melbourne, Australia

HITSI Application
By
Christopher Tivendale

Document: Thesis
Tutor: Dr. Seng Loke
Date: 21 May 2007

Abstract

Sensor based human interaction with computer applications is an emerging field that promises to change the way we have traditionally communicated with computer systems. Many applications have been designed using sensor based human interactions with different methods and goals for success. In this paper I look designing an application that uses human interaction through sensor interpretation (HITSI) to control a third party application. The HITSI application design model and decisions are discussed including a prototype implantation of the HITSI application. The HITSI Application prototype is analysed for performance and limitations. Users inexperience with the technology used and a lack of standards are two of the limitations that guide the look at where future work can be aimed in human interaction through sensor interpretation.

STATEMENT OF AUTHORSHIP

I certify that the attached document is my original work. No other person's work has been used without due acknowledgement in the text of this document.

Except where reference is made in the text, this document contains no material presented elsewhere or extracted in whole or in part from a document presented by me for another qualification at this or any other institution.

FULL NAME: Christopher Tivendale

DOCUMENT: HITSI: A Human Interaction through Sensor Interpretation Application

SIGNED:

DATE: 21 May 2007

Acknowledgment

This project would not have been possible without the fantastic support and patience provided by my supervisor Dr Seng Loke who has been a pillar for guidance and clarity in my many times of confusion. Also thanks to the staff of the Computer Science and Engineering Faculty of Latrobe University Bundoora who have provided both equipment and assistance promptly and with smiles. And finally Thankyous also go to my family and friends who have with their love and belief both supported and pushed me into achieving my best.

Table of Contents

Table of Figures

Chapter 1 Introduction

Computers have become a natural part of modern society's daily lives in the last ten years. From work in the office, playing games at home, or checking and writing e-mails while on holiday humans are always interacting with computers. These interactions have been mainly through a keyboard and mouse in the past, but with advances in mobility, price and technology in sensors a new form of interaction can soon take place. Human based sensor interaction!

In the not so distant future it could be possible to command and interact with a computer system, not with a keyboard and mouse but by voice, gestures and everyday behaviour. My thesis looks at how far these ideas and technologies have come and what issues are relevant to these design and development of a human based sensor interaction computer application.

1.1 Aims

The aim of this thesis has been to develop a prototype application (referred to as the HITSI Application) that is capable of interpreting sensor based human interactions to control a third party computer application.

The design of the prototype must allow for users to modify configuration options that relate to both sensor interpretations as well as third party application actions. Sensor Interpretations occur when multiple sensor inputs are used together to give a single meaningful outcome for the HITSI Application.

To allow for sensor interpretations, methods to describe sensor inputs and resulting interpretations need to be designed. These methods not only need to fully describe sensor interpretations but also be designed in a way that users can understand and modify.

Multiple third party applications need to be able to make use of the applications sensor reading and interpretation results. Users should also be able to have the same sensor readings interpreted differently for different applications. Interpreting sensor readings involves grouping all the sensor readings in a certain time period together and then looking up this pattern of sensor readings and seeing if it is included in a set of predetermined interpretations.

In the event that there is multiple sensor interpretation ambiguity a set of rules needs to be implemented that can discern which interpretation is in fact the correct interpretation. These rules could potentially make use of a scoring system, priorities, first match or a mixture.

1.2 Approach

To implement sensor activity as a form of input I first needed to decide upon which sensors I would use. I needed to have sensors that would connect easily to a computer system and have an API that would make communication between sensor and computer easy. I decided upon the Phidgets interface Kit as it met these conditions.

To effectively use the sensor inputs provided by the Phidgets interface kit a set of pre-determined rules needed to be designed. These rules would be used to determine that the sensor input received was correctly interpreted and transformed into a resulting action used by HITSI Application. These rules not only need to be stored but also need to be easily modified upon the users discretion. As such I decided to store these "rules" in an external location that is read into the HITSI Application at execution time. XML was Ideal for this use and is the structure I have used to store these "rules".

In deciding upon a development Platform for the HITSI Application the above considerations need to be addressed and implemented using the chosen platform. Microsoft C#.net provided a platform that incorporated XML and was capable of supporting the Phidgets Interface Kit API.

In the case where multiple sensors are used for several different action interpretations a set of rules needs to be applied to decide which sensor interpretation is the correct or expected choice. Methods such as first detected, scoring system and priorities were considered.

1.3 Overview

The Literature Review (Chapter 2) looks at several sensor based human interaction techniques and systems including standards and architectures that have been designed previously. Topics that are covered include Pervasive/Ubiquitous Computing, Multimodal Interaction, Smart Spaces and Context Awareness.

The HITSI Application Model (Chapter 3) looks at outlining a design that will meet the requirements and aims outlined in sections 1.1 and 1.2. It makes use of UML diagrams including a use case, block diagram and flow chart as well as discussing issues that arose from this design and possible solutions.

The HITSI Application Implementation (Chapter 4) further enhances the ideas presented in the HITSI application Model chapter by introducing a low level design that makes use of a class diagram, sequence diagrams and a component diagram. The section also looks at the technologies used to create the HITSI Application prototype including the database and its structure.

The next chapter is the HITSI Application Evaluation (Chapter 5). This chapter Analyses the performance, accuracy and soundness of the HITSI Application Prototype. The section also covers a real world example of where the HITSI application could be used and describes some of the limitations that exist for the HITSI Application.

The Conclusion (Chapter 6) summarizes the ideas, design and solutions discussed in this paper. It also looks into what developments can be produced for the HITSI application in the future.

Chapter 2 Literature Review

Sensor based human interaction with computer applications is an emerging field that promises to change the way we have traditionally communicated with computer systems. Many applications have been designed using sensor based human interactions with different methods and goals for success. In this literature review we look at several sensor based human interaction techniques and systems and any standards or architectures that have been proposed or implemented including any shortcomings in functionality or design that has not been approached.

This literature review discusses previous attempts and successes in developing a sensor based human interaction application. The topics Pervasive/Ubiquitous Computing, Multimodal Interaction, Smart Spaces and Context Awareness are all explored and contain the general theme of sensor based interactions with computer applications.

At the conclusion of this report a viable architecture and framework should be materializing for use in developing a sensor based human interaction system.

The rest of this literature review is organized as follows:

Pervasive / Ubiquitous Computing (Section 2.1)

A goal of many computer systems is to achieve a sense of invisibility to the user, yet include human interaction. This section looks at what Pervasive/Ubiquitous Computing is, what are the goals when designing a Pervasive/Ubiquitous computing environment, some examples of

Pervasive/Ubiquitous computing and what are the issues associated with Pervasive/Ubiquitous computing.

Sensors (Section 2.1)

Sensors are a key element in Human Interaction with Computer applications. The Sensor section gives a description of sensors and their uses as well as looking at some of the issues associated with sensors in a computing environment.

Multimodal Interaction (Section 2.3)

Humans often interpret themselves with many forms of interaction such as speech, facial gestures, arm movements and so on. Using more than one of these interaction techniques with a computer at one time is a Multimodal Interaction. This section defines the what, why and where of multimodal interaction, as well as taking a look at different Multimodal Architectures in a computing environment.

Smart Spaces (Section 2.4)

Smart Spaces make use of multimodal interaction methods, architectures and techniques to achieve there computing goals. This section defines what a smart space is, illustrates an example and looks at where smart spaces are headed in the future

Context aware Computing (Section 2.5)

This section describes what context and context aware computing is as well as outlining several architectures to achieve context aware computing.

Discussion (Section 2.6)

Ubiquitous/pervasive computing, Smart spaces and Context Awareness all share similarities and differences. The discussion looks at these and also what functionality is not included in the current approaches.

Conclusion (Section 2.7)

A short description of a system that includes functionality that has not been attempted in existing systems that is to be implemented in the future is included in the conclusion.

2.1 Pervasive / Ubiquitous Computing

Mark Weiser the person accredited with creating the idea of ubiquitous computing once wrote the following when describing the trends in computing [1]:

“Mainframe:	Many people share one computer
Personal Computer:	One person, One computer
Ubiquitous Computing:	Many computers serve each person”

In [2] he defines Ubiquitous Computing further with “Ubiquitous computing is the method of enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user.”

2.1.1 Goals of Pervasive Computing

The goal of designers of ubiquitous systems is to create a system that meets the following criteria:

- All computation and computer processing is to remain unseen by the user
- All interactions are to remain unobtrusive
- The hardware components are to be completely emerged in the environment
- The system must be intuitive
- The system must always available.

Along with these goals, ubiquitous computing aims to create an entirely new interface with computers.

The challenge in ubiquitous computing is hiding the computer and yet making it vastly more intelligent. Ideally the ubiquitous system would be totally unnoticed by the user. The system would simply operate and make a user's life simpler without any active interaction.

2.1.2 Ubiquitous Computing in Action

A successful attempt at ubiquitous computing is the Classroom 2000 project developed by the Future Computing Environments Group in the College of Computing and GVU Center at Georgia Tech [3]. The Classroom 2000 tackles the problems involved in information intensive lectures. While undertaking lectures in topics in Computer Science and Engineering it is not uncommon to switch between a slide show, the internet and program examples. In such an environment it is often very difficult to take full notes on what has occurred during the lecture. The classroom 2000 monitors a lecture's progress and records all information used by the lecturer in chronological order.

In the classroom 2000 project lectures go on as normal without anyone aware of the computer taking recordings. In fact the objects in the room are not exactly what they appear to be. As you can see in figure 1 below the Classroom looks like any normal classroom, however objects such as the whiteboard are in fact interactive surfaces where markings made on these objects are recorded by the computer system. In this respect the Classroom 2000 is able to achieve apparent transparency.

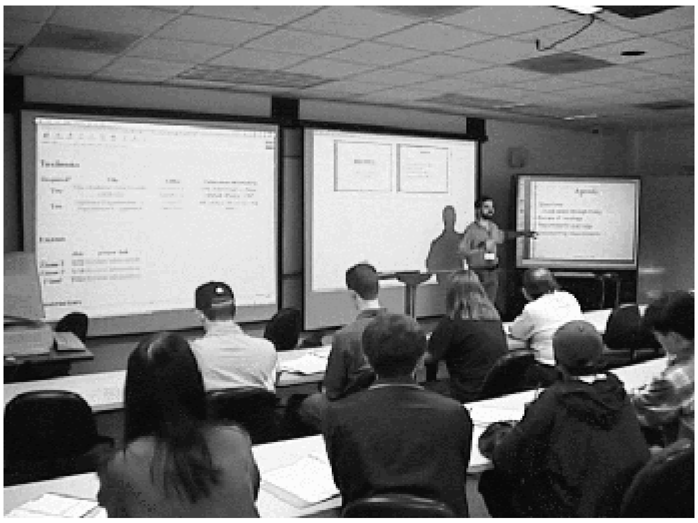


Figure 1: Classroom 2000

2.1.3 Issues with Ubiquitous Computing

Security Risks

The amount of information that a Ubiquitous Computing system can record and store is rapidly increasing. It will not be long before ubiquitous systems are recording things like TV watching habits, Driving behaviours, emotional state and even medical health. Such personal information will almost certainly attract those who would use such information to the benefit of themselves or their interests. The information gained from a ubiquitous system attack could be used for invasive personalized advertisements or even by stalkers or blackmailers who use the ubicomp system to monitor a user's whereabouts as a form of surveillance.

The invasion of privacy and online tracking among other issues on the Internet has been a large problem since the Internet's creation. The same vulnerabilities in an ubicomp system would produce greater harm because of the unprecedented amount of data used [4].

Personalization

Ubiquitous systems provide the possibility to personally identify each user inside of the operating environment. Current designs and implementations of Ubiquitous systems have not however included the possibility of allowing for Personalisation options. An example of a personalization option could be the vocalized greeting from a ubiquitous system when a user enters the environment for the first time each day. A standard greeting may consist of "Greetings <username>" however a personalized greeting for user "Patrick O'riely" may consist of the vocalization of "Greetings and Salutations Patrick O'riely". Individual personalization could potentially decrease a new users "fear" of a ubiquitous system and allow for more widely accepted use and implementation.

Configuration Options

So far each Ubiquitous system has been designed for one specific purpose and each human interaction has a specific predetermined meaning. There is potential in an ubicomp system to introduce dynamic configuration of interpretation options. An example of this sort of configuration options could be an arm raised is interpreted as turn On the lights, however a user with a disability that makes it impossible to raise an arm may wish to use a vocalized command. The design and implementation of this feature could be incorporated into new ubiquitous environments.

2.2 Sensors

According to Wikipedia "A **sensor** is a physical device or biological organ that detects, or *senses*, a signal or physical condition and chemical compounds."[1]

In particular we are interested in the physical device that detects signals and/or physical conditions. The Free Dictionary defines a sensor as "A device that measures or detects a real-world condition, such as motion, heat or light and converts the condition into an analogue or digital representation."[2]

2.2.1 Sensors in Computing

Sensors are used in computing for many varied purposes. Sensors are being used to cook your toast, water your plants, feed your pets and even monitor your movements. An example of sensors working together is in a ubiquitous computing environment. Figure 2 illustrates how some Japanese cities have used the T-Engine [5] which uses hundreds of tiny sensors embedded in pavements, street furniture and roads to allow for tourists and the handicapped to effectively navigate their way around these cities.

[1] http://www.wikipedia.org/
[2] http://computing-dictionary.thefreedictionary.com/sensor

Figure 2: Sensors in use in a Ubiquitous Computing Environment in Japan

Sensors can also monitor conditions in an enclosed environment such as a conference room, lecture theatre or home. Connecting these sensors to a computer system will allow the system to act in response to human actions and environmental conditions inside that environment.

Sensors can vary largely in size and cost. The figure below shows two sensors that are currently in use in ubiquitous environments.

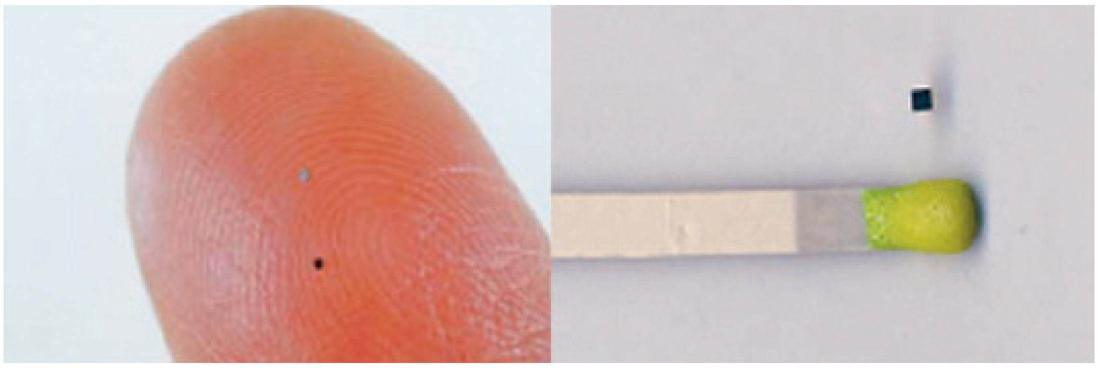


Figure 3: Hitachi's mu-Chip and Toppan's T-Junction ucode Tags

Sensors are also released as kits, which consist of a selection of sensors controlled through a central piece of hardware. An example of one of these is a Phidget Interface Kit [6]. A Phidget Interface Kit consists of sensors such as slide sensors, touch sensors, ir-distance sensors, motion sensors, light sensors and more, which can be connected and disconnected through a central control board. This control board can be interfaced to a computer system through a USB connection. The Phidgets Interface Kits include all the necessary software and hardware to enable a designer to begin development immediately.

2.2.2 Sensor Considerations

Some mobile sensors being developed have limited power outputs. These power constrictions limit the effective range of these mobile sensors to 30cm or less. In systems that are attempting to achieve a ubiquitous environment, requiring users to move to within 30cm for identification may destroy the 'invisibleness' of the environment.

Another issue to consider when developing computer systems that use sensors is what level of generalization and stability is required to meet the goals of the software design. If a system is to be designed to enable sensors to be added and removed during runtime and to reach a high level of generalization then the software components of each sensor must be separate from the hardware components to allow for design simplification and also to reach desired generalization. The Phidgets interface kit mentioned earlier is released with an Application Programming Interface (API) to enable communication to the sensor through a variety of programming languages and is a good example of hardware and software separation in sensors.

2.3 Multimodal Interaction

In 1980 Bolt created the first known Multimodal Application. In this application he used the ‘put that there’ interface, where a user could point to a touch pad and speak commands. While very primitive this application led the way in multimodal interfaces.

Multimodal interaction consists of two or more inputs combined together in a way that produces a calculated response in terms of a system output. Such inputs can consist of speech, pen, gestures, touch and any other sensor that is connected to a system.

2.3.1 Successful Multimodal Interaction

According to T.V.Raman in [7], there are five main criteria for success in a multimodal system, which are

- multiple modalities need to be synchronised

  This is the synchronization of inputs in a multimodal system.

  This is especially important when combining temporal inputs such as voice with spatial inputs such as visual interaction. Without synchronization the overall usability of the interface is diminished if not non-existent.

- Multimodal interaction should degrade gracefully

  Graceful degradation is where a system offers less service (such as dropping vocal inputs in a noisy environment) instead of failing completely. It is important to maintain graceful degradation as this mimics human interaction where a user’s needs and abilities can change over time.

- Multiple modalities should share a common interaction state

A common interaction state allows all interactions to communicate and interpretations to be reached from all interactions.

- Multimodal interfaces should be predictable

Multimodal interactions can often achieve the same results with two different modalities. A multimodal interface should be designed in such a way that the modalities available are intuitively recognised by the user. An example would be highlighting an input box for pen input and vocalizing that speech recognition is an accepted input.

- Multimodal interfaces should adapt to users environments.

A multimodal interface should be able to adapt to the users needs and abilities, bandwidth and constraints applied to the user by the environment (Driving - hands free)

2.3.2 Multimodal Interaction Benefits

The following is a list of benefits that Multimodal Interaction can offer [8]

- Human perception and communication skills allow for multiple modes of interaction and in some cases multimodal interaction is a more natural interaction compared with single input interaction
- The Benefits of each individual mode can be highlighted and used
- A selection of modality can occur depending on context, environmental conditions and user preference.
- Recognition errors received through an individual sensor can be analysed with other inputs and these errors can be eliminated or corrected providing the system with a correct reading and providing an overall more stable and robust system. This process is called “mutual disambiguation”.

- Multimodal Applications can be used by people with a wide variety of impairments. i.e. visually impaired can use the voice component, hearing impaired can use the visual component and the situationally impaired[3] can use vocal or gesture recognition components.

2.3.3 Multi Modal Systems in Operation

Multimodal systems are applied to many applications such as telephone services, security systems and Virtual Reality for Simulation and Training [9].

The practical use of sensors in multimodal applications is large and varied. Currently a mix of Pen and Speech based input is the most common and mature sensor input combination, however video inputs in such applications as security systems and Smart Rooms or offices are also are becoming more common place. When using a video input several different recognition techniques can be used. Such things as gesture recognition, lip reading, and movement tracking have all been developed. These technologies are able to help those with disabilities in the form of monitoring their situation and alerting those who need to be notified of any dangers, and also as a means of communicating with a computer system that they would otherwise not be able to do.

[3] "Situationally impaired" is when a person that can normally complete an action but cannot at the current moment because of their situation, e.g. wearing gloves or in a noisy environment.

2.3.4 Input Modes

Multimodal interactions can be further defined by considering what types of input modes the interaction uses.

Active Input Modes

These input modes are interactions that are intentionally initiated by the user as a means to interact with the computer system. An example of this could be in the form of a vocal command or keyboard input

Passive Input Modes

A passive input mode is a user action or response that can be recognized by the computer. Such actions are passively monitored and can include inputs like manual gestures or facial expressions.

2.3.4.1 Multimodal Interfaces

A multimodal interface can consist of only passive inputs, only active inputs or a variety of both. Besides the general Active or Passive Multimodal interface where only active or passive modes are present two interfaces which can incorporate both are mentioned below.

Blended multimodal interfaces [10] consist of at least one passive and one active input mode that are recognized by the computer system. An example of this sort of interface is a system that monitors both speech and lip movement.

Temporally-Cascaded Multimodal Interfaces [10] can consist of only active inputs, only passive inputs or a mixture of both. What defines a temporally-cascaded multimodal interface is that modalities are sequenced in a temporal order. The information of an earlier input can determine the result of a later input. Such an example can include a vocal instruction then an action (such as pointing).

2.3.5 Multimodal Architectures

For systems designed to handle joint processing of input signals there are two subtypes of multimodal architecture to consider. These architectures are split into the ones that fuse signals at the feature level and those that integrate information at the semantic level.

2.3.5.1 Feature Fusion

Feature Fusion Architectures [11] fuse low level parallel signal input information in a Multimodal architecture. Feature fusion is considered more appropriate for Blended Multimodal Interfaces such as speech and lip movement. Speech and lip movement are often linked together as speech recognition correctness is increased with visual cues.

Disadvantages

Has a high computational cost when many sensors are reporting data at any one time.

Because feature fusion attempts to fuse all data when it arrives, even when data may not be needed resources are used, and as more sensors arrive more and more resources are used.

Because Feature Fusion only fuses parallel input signals, signals that are received in a scattered time period cannot be fused and hence interpretations of sensor inputs that require two or more sensor inputs that are temporally cascaded is not possible.

2.3.5.2 Semantic Fusion

Semantic Fusion Architectures [11] integrate semantic data from parallel signal input recognisers in a Multimodal architecture.

Semantic fusion can make use of an adaptive agent architecture. The following is an example of the flow of input in a semantic fusion operation.
Each input comes through a modality agent such as a gesture recognition agent, and gets passed to the Facilitator which time stamps the input and passes it onto another agent for further language processing such as a gesture interpreter. This then gets passed back through the facilitator and onto the multimodal integration agent. The agent then decides how long to wait (temporal window) for more inputs before fusing the meaningful fragments of information together and passing through the facilitator again and onto the application which can then use this information.

Benefits

- Semantic Fusion has separate recognizers for each single modality.

  This means that all modalities can be interpreted with their own single recognizers separately and integrated together without any more interpretation needed.

- New recognizers can be used, as interpretation is done separately for each input stream.

Unlike Feature fusion Semantic Fusion only fuses input signals after they have been recognized and interpreted at a low level by their individual recognizers. Because these recognizers are not built into the fusion components, individual recognizer components can be added without the need to replace the semantic fusion component or any of the other recognizers.

- Flexible Asynchronous architecture.

  Because fusion is completed separately then sensor input and low level interpretation components such as new recognizers or new fusion components can be added to the architecture without any interruption to the design or operation of the system.

Disadvantages

- Decisions are subject to temporal and semantic constraints, i.e. speech and gesture overlap or gesture precedes speech by less than four seconds or speech does not precede gesture. The constraints applied mean that any interaction that does not strictly fit into a temporal window will not be interpreted properly and the usability of the system suffers.

2.3.5.3 MicroTemporal and MacroTemporal Fusion

MicroTemporal and MacroTemporal fusion are similar to Semantic Fusion but separate the simultaneous input stream fusion from the temporally cascaded input stream fusion. [12] MicroTemporal Fusion is used primarily to combine information that is produced by inputs in parallel or overlapping times and MacroTemporal fusion is used to integrate data that has been received without an overlap but within a temporal time window.

2.3.6 MultiAgent Information Flow Architectures

Whilst handling the flow of multiple inputs the flow of information processing may also be considered as an architecture. Multiagent architectures are used to model these systems. The two Multiagent options include the open Agent Architecture and the Adaptive Agent Architecture.

Several authors in [13] describe agents as “conceptual entities that perceive and act in a proactive or reactive manner within an environment where other agents exist and interact with each other based on shared knowledge of communication and representation.”

For the purposes of the examples below an agent is a component that is able to register its service in an acceptable form, be able to speak the interagent communication language, and provide functions that are standard to all agents such as the method of data storage and retrieval.

2.3.6.1 Open Agent Architecture

The open agent architecture was designed to provide developers with the option to support multiple software packages in an agent architecture while shielding the developer from communication issues.

Each component in the open agent architecture can be developed in a different language but is wrapped in its own software that is able to communicate in a standard language. The component along with the surrounding communication layer is called the agent.
These agents interact directly with other agents and components with which they have information.
Below is a diagram of a typical Open Agent Architecture

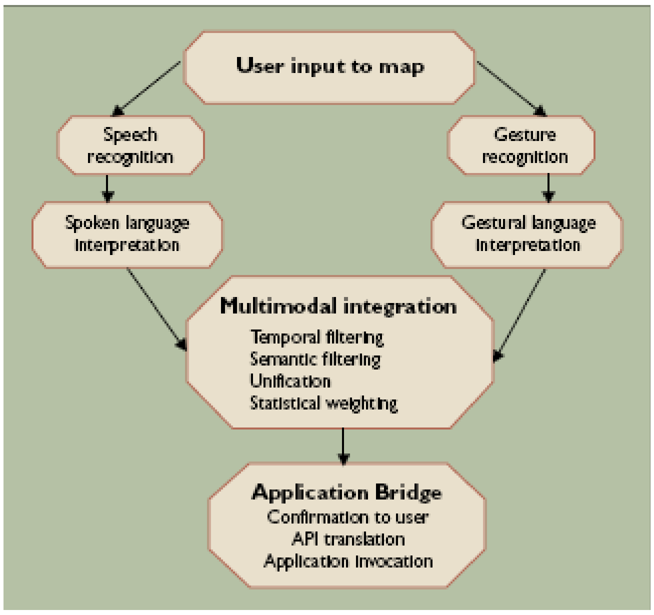


Figure 4: Typical Open Agent Architecture

The advantage of this communication is that there are no intermediaries. With no intermediaries communication between sensors and the Application can be accomplished at a much faster rate. The flip side of no intermediaries is that the Application relies very strictly on the agents operating properly. If an agent is no longer accessible there is no way for the application to detect this and correct the problem and the system can effectively fail.

2.3.6.2 Adaptive Agent Architecture

To overcome the failure issues of Open Agent Architectures a facilitator or Middle Agent is introduced. This allows the Open architecture to become adaptive to finding, using, managing, and updating agent services. Below is a diagram consisting of all the same basic elements of the Open agent architecture in figure 2 but with a facilitator now. The facilitator now handles all communication between agents. This allows for components to be added and removed during runtime. Therefore if the gesture recognition agent was lost somehow (perhaps poor network connection) then the facilitator would now and be able to communicate this information to any interested agent. Similarly if it was once again available (perhaps the network connection was re-established) the facilitator is able to reintegrate the gesture recognition and inform all agents of its availability.

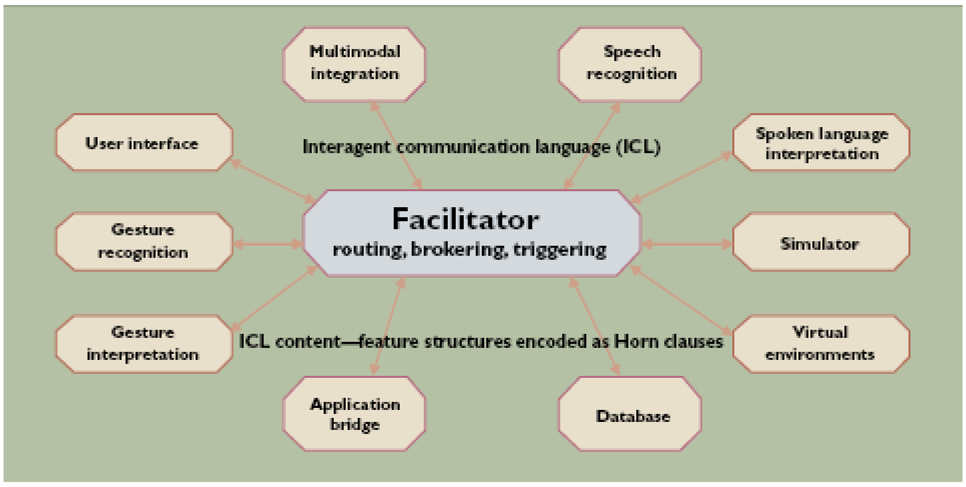


Figure 5: Adaptive Agent Architecture

2.3.7 Comments

By incorporating a fusion technique that meets the criteria of the system and an architecture that will support the functionality required into your system you can effectively meet the guidelines for a successful System with multimodal Interaction.

By using an adaptive agent architecture agents can be programmed in different languages and exist on different systems, agents can be added or removed at runtime, agents can cooperate on tasks in parallel and ranking of tasks can be justified.

2.4 Smart Spaces

Smart Spaces are environments equipped with sensors that interact with a computer system that can interpret and react to actions and changes in its environment. These areas can be limited to individual buildings or rooms and can potentially be as large as entire cities and eventually perhaps the world. Currently sensors are being built into all types of everyday objects. Desks, doors, cars, shoes and cookware can all contain sensors. While not individually "smart" these devices can potentially be grouped together in to a larger network of sensors that can react and reason, sending signals to attached devices, printers, people or mobile phones.

2.4.1 Smart Space Applications

Smart spaces are able to use sensors to track people's movements, hand gestures, recognize vocal commands, understand body movements, interpret moods, lip read [14] and understand many other interpretations of inputs to make calculated decisions based on the information provided to it. While some of these technologies are fairly mature and have been commercialised, many others are still in the design and testing phases.

Currently Smart spaces are being used to control Virtual Reality environments, aid those with disabilities both at home and at work, and to control computer systems without the use of mice or keyboard.

2.4.2 The Intelligent Classroom

A practical example of a smart space is a smart classroom. Inside such a smart room sensors would monitor teacher's actions and record necessary information without being issued any commands as well as predict what the teacher is going to do next and automatically do this. Such an example would be when the teacher reaches the end of a slide the room correctly identifies this and proceeds to the next slide without any command from the teacher. A project called the Intelligent Classroom (Different from Classroom 2000 mentioned earlier) has taken steps towards this end [15].

The Intelligent classroom makes use of two sensors, namely a microphone and video camera. Through the use of speech recognition and gesture recognition the Intelligent Classroom can correctly identify the current progress of a lecture and automatically progress through slides as the lecturer finishes each one. The Intelligent classroom is also not only able to video a teachers lecture but also to zoom in where the teacher points using gesture recognition and also to focus on white boards and projector images.

2.4.3 Smart Spaces in the Future

As the technology involved in smart spaces become more widely used and costs decrease smart spaces could eventually appear in almost every room you enter. Around the house tasks like vacuuming could use gesture recognition to complete the job, vocalized ordering of shopping and even forms of security and intrusion detection could be handled by a smart space. Smart spaces offer an exciting future of possibilities that are only just beginning to be explored.

2.5 Context Aware Computing

The use of context in computing is not new and yet there is still no exact definition. Context has been classified into Physical and Cultural contexts by Benerecetti[16], Physical Environment, Human Factors and Time by Schmidt[17], Physical Environment, the User Environment and the Computing Environment by Lieberman and Selker [18] and many other different but similar definitions. In all of these examples an attempt is being made to translate the definition “that which surrounds and gives meaning to something else” into a set of rules and classifications that can be applied to a computing environment.

Abhishek Singh describes context awareness in [19] as “the state wherein a device or software program is aware of the environment and performs productive functions automatically, without requiring explicit control or notification.”

2.5.1 Context Aware Computing Progress

Of all the sensor based human interaction topics mentioned in this review Context awareness is the least mature. For years context aware systems have been simply designs and prototypes. Several applications model certain aspects of context awareness such as the projects developed by the future computing environments group [20] however a fully operational context aware system that that is able to model the entire environment is yet to be developed.

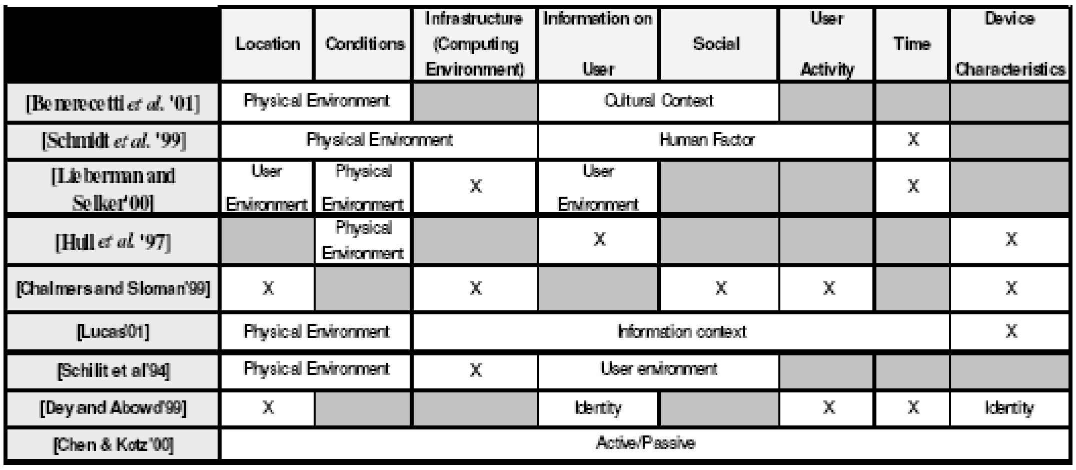
2.5.2 Frameworks

When proposing to use context aware computing a few decisions need to be made. The first is how you will model the environment you are in (Context Model), and the second is what software architecture you will use to allow the context model to be fully utilised in your software (Framework).

2.5.2.1 Context Models

A context model describes how elements interact and influence user's intentions when undertaking activities in an environment

Manasawee Kaenampornpan [21] uses the following table in his paper to outline similarities and differences in many recognized approaches to defining a context model.



	Location	Conditions	Infrastructure (Computing Environment)	Information on User	Social	User Activity	Time	Device Characteristics
[Benerecetti *et al.* '01]	Physical Environment		▨	Cultural Context		▨	▨	▨
[Schmidt *et al.* '99]	Physical Environment			Human Factor			X	▨
[Lieberman and Selker'00]	User Environment	Physical Environment	X	User Environment	▨	▨	X	▨
[Hull *et al.* '97]	▨	Physical Environment	▨	X	▨	▨	▨	X
[Chalmers and Sloman'99]	X	▨	X	▨	X	X	▨	X
[Lucas'01]	Physical Environment		Information context					X
[Schilit et al'94]	Physical Environment		X	User environment		▨	▨	▨
[Dey and Abowd'99]	X	▨	▨	Identity	▨	X	X	Identity
[Chen & Kotz'00]	Active/Passive							

Figure 6: Context Model Approaches

The eight elements common to these approaches are clearly not exclusively needed. Benerrectti address only four of the elements (Location, Conditions, Information on User and Social) and yet Schmidt addressed the same four elements and 3 others (Infrastructure(Computing Environment), User Activity and Time) 2 years earlier. There is clearly some ambiguity in how much information is needed to describe a situation efficiently and effectively.

To translate these descriptions into a formal that is understandable to a computer system a mark-up language can be used. XML or OWL are great examples of potential mark-up languages to implement your context model.

2.5.2.2 Context Framework

Once a suitable Context model has been decided upon a context aware framework needs to be approached. As in much of the information relating to context in computing there are several approaches to context aware frameworks.

When deciding upon a context framework there are several criteria that should be met in order to properly deal with context [12] .

1) Separated Context Acquisition and Use

This statement allows for applications to use contextual information without the need to understand how contextual information is obtained by a sensor.

2) Context Interpretation

This is the clear interpretation of context information. Context should be interpreted at the lowest possible layer of the application.

3) Transparent Distributed Communications

In a context aware system not all sensors may be directly connected to the computer that is running the context aware application. Therefore communications need to be made between the machines with the sensors and the machine where the context aware application is located. These communications need to be transparent to both the sensors and the Application. This transparency means that a designer need not worry about building a communication framework, and that the overall design of the system is simplified.

4) Constant Availability of Context Acquisition

Context aware applications need to access the components in the system that provide contextual information when the application requires it. The application itself cannot continuously instantiate the components. This leads to the fact that a component that acquires contextual information must be executed separately and independently from the context aware application. However as the Application may need data at any time the components must be available at all times.

5) Context Storage and History

The use of Context History is to allow an application to observe trends and to predict future context values. With history such analysis could not take place. A context aware architecture should attempt to support storage of contextual information at the lowest level of detail possible, this allows applications the best chance of predicting and interpreting correct results.

6) Resource Discovery

A context aware architecture should support the possibility to dynamically discover what sensors and contextual information are available to it at any point in time.

Below are three Architectural approaches that attempt to successfully deal with context. They all use sensor based interactions as the main form of communication between the user and context aware system.

2.5.2.2.1 The Context Toolkit

The Context toolkit makes use of the five components listed below to meet the requirements of a successful context aware architecture

Context Widgets

A context widget is a software component that allows an application to obtain context information about its environment. Context widgets provide the means to fulfil requirements one, two and four as listed above.

Interpreters

Interpreters operate by combining context information together to interpret new context information that is of a higher level of abstraction. An example of this interpretation could be two geographical coordinates which can be translated into an address. Context Interpreters achieve a level of abstraction that satisfies requirement three above.

All context interpreters have the same communication interface. This allows all widgets, applications and aggregators to communicate context information to the interpreters.

Aggregators

Aggregators collect multiple context inputs that are related and store this information in a common location. Aggregators are used to allow applications to request logically linked contextual information from a centralized location rather then having to communicate with each widget in turn to obtain a context input.

Aggregators also allow applications to query for updates, access stored context data and also inform applications of context changes. Aggregators provide a further level of separation between context gathering and context uses.

Services

"Services are components in the framework that execute actions on
behalf of applications." [12]

A context service acts similar to a widget except that it handles outputs instead of inputs. The context service allows a separation between controlling an action in the environment and the application. Such outputs to the environment are called actuators.

Discoverers

The role of a discoverer is to keep track of what components are available to an application at any time. This includes a list of all widgets, interpreters, aggregators and services that are available.

When a new component is installed it is required to report to the discoverer information on how to communicate including language, protocols and machine hostname. Specific information relevant to the component must also be reported such as what type of context a widget can provide and interpreters need to provide what sort of interpretations they can handle.

When a component is no longer available it is the discoverers responsibility to detect this and report it to the relevant and affected components.

Figure 7 on the following page shows the interactions of an example configuration of the context framework components.

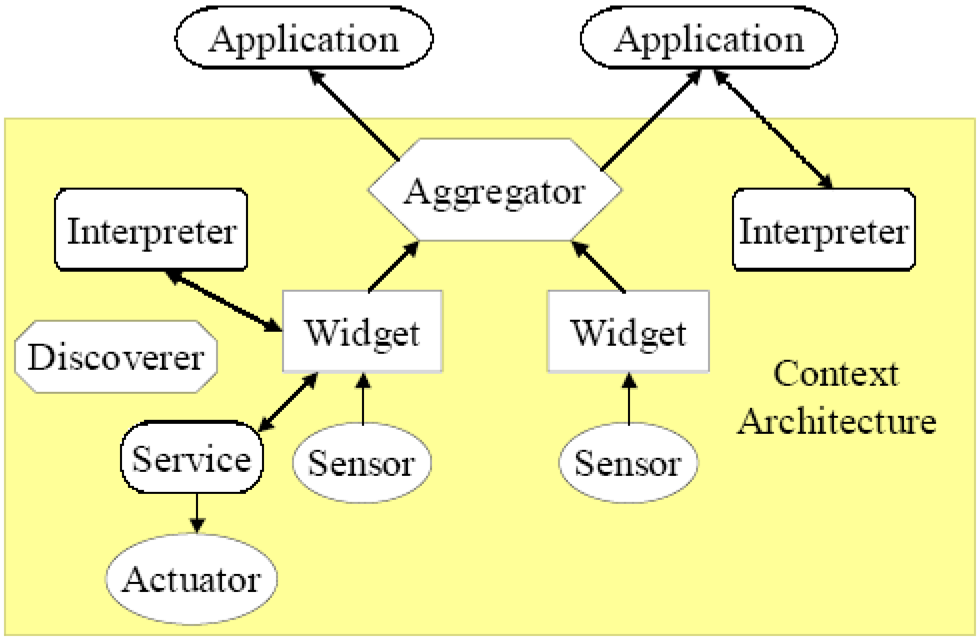


Figure 7: Context Toolkit Architecture

2.5.2.2.2 Context Broker Architecture

The Context Broker Architecture or COBRA is an architecture that is primary designed to support context aware systems in smart spaces. The main idea behind the architecture is the intelligent agent referred to as the Context Broker located as a central entity that maintains and shares context model information with agents, services and devices. The context broker is also able to provide privacy for the users as it is the central component of the architecture it can enforce privacy rules [22].

COBRA uses the web ontology language (OWL) and RDF to define ontology's of context. Ontology is a formal description of concepts in a certain domain. Effectively COBRA uses OWL to implement the context model and also to communicate contextual information between agents. In an example where a GPS is able to supply context data that “My car is near a supermarket” an OWL description of this situation could look as follows.

```
<socam: Car rdf:ahout=”MyCar”>
<socam:hasNearbyPlace rdf:resource=”#Supermarket”/>
</socam:Car>
```

Almost all interferences of contextual information are processed in the context broker. In this respect greater control of information is introduced over the context toolkit which has many components all exchanging communications. The context broker agent consists of several components which allow for the understanding of context.

Figure 8 contains a Context Broker Architecture. As seen in the diagram the context broker is able to receive contextual data from many sources and fuse this data into information, which can then be shared with other computing entities.

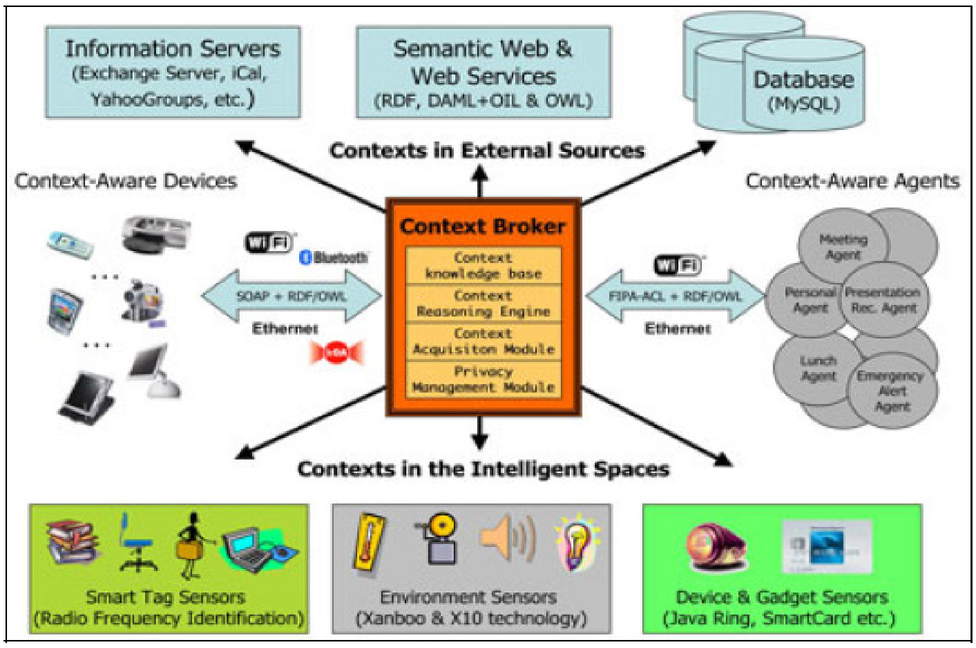

Information Servers (Exchange Server, iCal, YahooGroups, etc.)
Semantic Web & Web Services (RDF, DAML+OIL & OWL)
Database (MySQL)
Contexts in External Sources
Context-Aware Devices
Context-Aware Agents
Context Broker
Context knowledge base
Context Reasoning Engine
Context Acquisiton Module
Privacy Management Module
SOAP + RDF/OWL
Ethernet
Bluetooth
FIPA-ACL + RDF/OWL
Ethernet
Meeting Agent
Personal Agent
Presentation Rec. Agent
Lunch Agent
Emergency Alert Agent
Contexts in the Intelligent Spaces
Smart Tag Sensors (Radio Frequency Identification)
Environment Sensors (Xanboo & X10 technology)
Device & Gadget Sensors (Java Ring, SmartCard etc.)

Figure 8: COBRA Architecture

2.5.2.2.3 Service Oriented Context Aware Middleware (SOCAM)

SOCAM like COBRA bases its design around ontology. It also uses Owl as the mark-up language to model these ontology's. However SOCAM is similar to Context Toolkit in that it has multiple components to interpret and share context data with. According to its original designers SOCAM aims to achieve a rapid prototyping of context-aware services in pervasive computing environments [23].

SOCAMS five main components are Context Providers, Context Interpreters, Context Database, Service Location Service, Context aware mobile services. The five components and there interactions are shown below in figure 9.

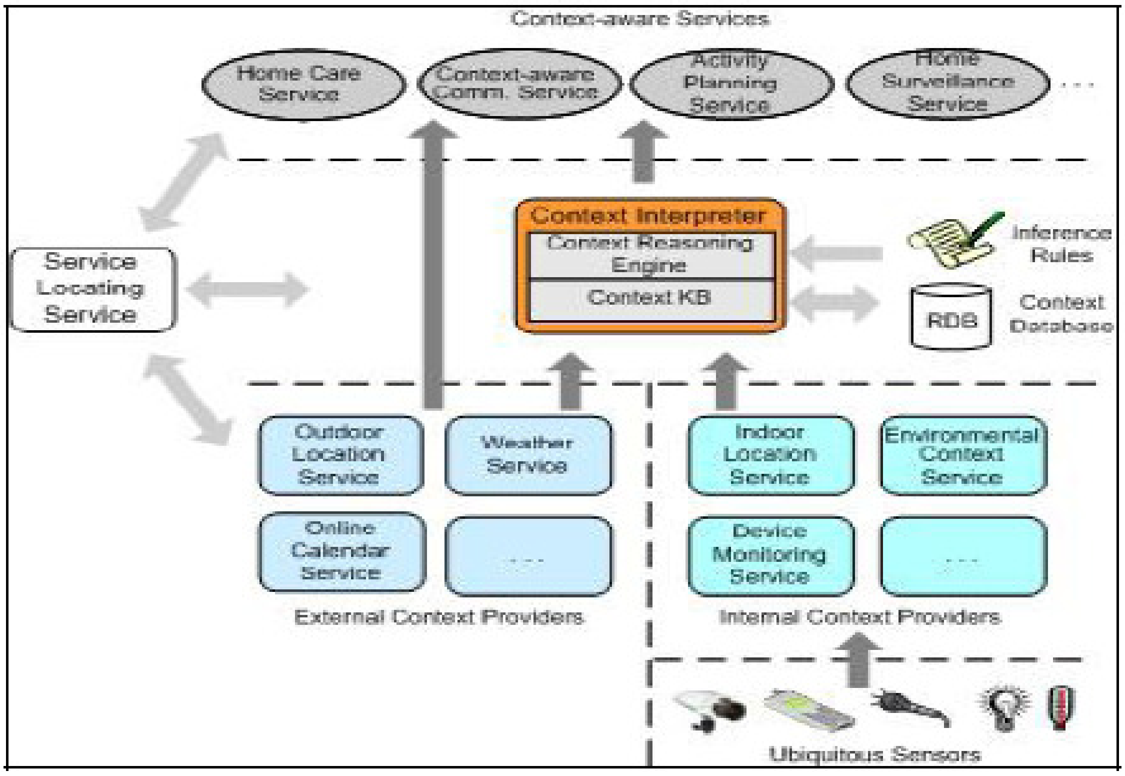


Figure 9: SOCAM Architecture

Context Providers

Context providers act very similar to widgets in the context toolkit. However All Context Providers will translate context received from outside sources into OWL descriptions. These descriptions can be understood by any component in the SOCAM Architecture.

Context Interpreters

A context Interpreter has three main functions. It acts as a Context Provider in that it provides higher level context information to components that need the said information.

The Context Interpreter also contains at least one Context Reasoner. The context Reasoner is able to deduce context information from other direct contexts. In this respect the context Interpreter acts as an Interpreter in the Context Toolkit.

The third function of the Context Interpreter is to provide a set of API's to other components so that they are able to query, add, delete or modify current context knowledge. The Interpreter therefore provides the means for other components to access the Context Database.

Context Database

The Context Database simply stores context information using OWL descriptions. It can be accessed by any service that can communicate with the Context Interpreter.

Service Locating Service

The service locating service has the same responsibilities as the discoverer in the Context toolkit. The Service Locating Service achieves its results through the use of OWL expressions as well as an option to use service templates when trying to understand what context a context provider provides.

Context-aware Services

Context aware services are very similar to the actuator in the Context Toolkit. A Context aware service responds to the current context or changes in context. They are able to respond through queering a context provider or listening to updates provided by the context provider. Unlike the actuator in the context toolkit however a Context-Aware service is able to deduce what action it should take from context data. The Context-Aware Service is able to do this by use of written pre-defined rules that specify what methods and actions should be fired under certain context. This data is stored in a file and can be accessed and updated at any time including runtime.

Communicating

It should be noted that in the SOCAM approach individual service components can communicate in either push or pull modes. A push mode is where a service subscribes to a context provider. This subscription is for a certain piece of context information. When the context information changes, it is automatically sent to the subscribed services. A pull mode is when a service simply queries a context provider for context data.

2.5.2.3 Similarities and Differences

The above architectures all try to separate context task responsibilities into separate areas. SOCAM and COBRA use agent based architectures whilst the Context toolkit separates them into Components. As mentioned earlier the main difference between components and agents is that agents can be developed in different languages and wrapped in software that is able to communicate in a standard agent communication language, as well as alerting other agents of its presence and sharing common features such as the way data is stored while components need not have these properties. While the architectures above are not an exhaustive list of solutions they are the three of the most commonly discussed. Other architectures such as the Context Mediated Framework [24], Prottoy [25] and EasyLiving [26] are also established Frameworks.

Of all the architectures the Context Toolkit attempts the most generalization with no specified communication language, COBRA attempts to achieve higher levels of security and SOCAM is a less generalized version of the Context Toolkit as it has some componentisation but not extensive and it is also restricted to using OWL as its context description language.

The COBRA approach fails to meet the requirements for a successful framework as it does not separate the Context Acquisition effectively enough from the use of the context. Similarly the Context broker is not able to handle dynamic changes to sensor availability.

While the Context Toolkit seems to meet all the required criteria for a successful framework, the requirements for successful criteria may not be complete. The COBRA system while not meeting all requirements has introduced a new requirement, namely Security. While the generalisation in the Context Toolkit would allow for security to be implemented it has not been approached In the design. If the Context Toolkit were to include another component between the aggregators and the applications then this component could handle what sort of access applications have to certain information.

2.6 Discussion

The definitions included in sections 2.4 and 2.5 state that "smart spaces are environments equipped with sensors that interact with a computer system that can interpret and react to actions and changes based in its environment" and that "multimodal interaction consists of two or more inputs combined together in a way that produces a calculated response in terms of a system output".

These two ideas are similar by definition however Smart Spaces concentrate more on the physical environment, human interaction through sensors and intelligent computing decisions compared with multimodal interaction, which considers techniques to effectively use multiple inputs to produce valuable results.

Similarly Mark Weisers definition of ubiquitous computing [reference] "Ubiquitous computing is the method of enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user" highlights that while smart spaces concentrate on the physical environment similar to ubiquitous computing they concentrate more on direct active sensor interaction. Smart Spaces are simply a sub-field of Ubiquitous Computing.

Abehishek Singh's description of context awareness "the state wherein a device or software program is aware of the environment and performs productive functions automatically, without requiring explicit control or notification" highlights that context awareness is similar to pervasive computing in that context awareness incorporates passive monitoring of the environment and similar to smart spaces in that intelligent decisions are made from sensor inputs. Context awareness however focuses on modelling context into a computer system. Similar to Smart Spaces, Context Awareness is a subfield of Ubiquitous computing.

The architectures included in Multimodal Interaction and Context awareness contain many similarities. As the two diagrams in figure 10 below show the flow of information is very similar.

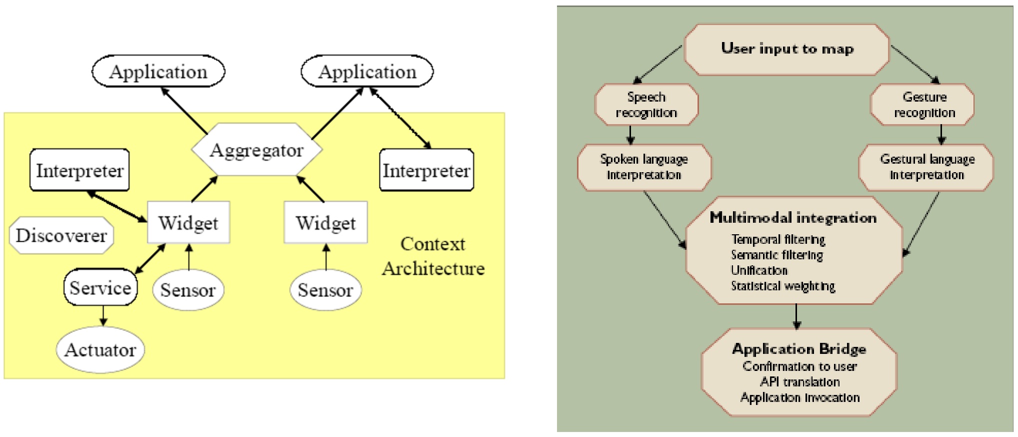


Figure 10: Context Toolkit Vs Multimodal Open Agent Architecture

Sensor data is interpreted then processed with other sensor data to provide valuable information that is then either passed to the application or accessed by the application.

The Widget components in the Context Toolkit architecture is a software component that allows an application to access context information about its environment. The Open agent architecture uses recognition agents to achieve the same results.

A context toolkit widget is able to communicate with an aggregator and interpreter in order to interpret and fuse context information retrieved form the environment. The aggregator also provides a centralized location for an application to retrieve logically linked contextual data. In an Open Agent architecture interpretation of individual sensors is handled by interpretation modules and the multimodal integration is performed in the multimodal integration agent. This agent also provides a centralized location for the application to retrieve multimodal information.

The context aware architectures however provide a means to model and store context information. Context aware and multimodal architectures by design support multiple fusion techniques designed for either architecture.

Whilst all the architectures contain similarities, several issues are not approached in any of them. The ability for a system to respond and act differently to individual users has not been investigated. This sort of personalization would require that each user in a smart space, Ubiquitous computing environment or context aware environment would need to be individually identified and monitored. While this feature would require large amounts of processing power as separate computations and decisions need to be made for each individual user it would be an impressive feature to implement in the future.

A similar feature would allow interpretation rules to be dynamically modified. This would allow an administrator to modify a systems interpretation of a human interaction at runtime to include different programs to be accessed or executed. In a smart space the original purpose of the system could be changed while still using the same framework. As an example a large conference room could be set-up in "conference" mode where the smart space was able to run PowerPoint, the lights and record proceedings or alternatively it could be set to "Party" mode where the system is able to determine if anyone is dancing and modify song selection accordingly, run party lighting by how many dancers and people present, and adjust music volume to match noise from users in conference room.

2.7 Summary

Sensor based human interaction in computer applications has advanced in many directions since 1980 when Bolt first developed his "put that there" interface. Science fiction movies even as recently as Minority Report in 2002 have introduced the idea of sensor based human interaction with computer applications as a normal part of everyday life. The reality of achieving this is not so far in the future. Many smart spaces and ubiquitous computing systems architectures and frameworks have been developed now to a point where they are out of the laboratories and are being used in the workplace.

Chapter 3 HITSI Application Model

This chapter outlines the design of the HITSI application. The information in this chapter outlines the issues that arose and solutions developed when designing a solution to meet the aims outlined in chapter one. The high level structure of the HITSI application is also proposed and explained including detailed explanations of how the HITSI application will achieve sensor based human interaction with a computer application.

3.1 Application Design Issues

The information contained within this chapter describes the issues and potential solutions that arose when attempting to meet the requirements and aims outlined in the previous chapters. Concepts including "Sensor Interpretations", "Sensor Interpretation Ambiguity" and Time Windows" are all approached.

3.1.1 Sensor Interpretations

Sensor readings placed into a computer application are meaningless, to become a meaningful input the sensor readings need to be interpreted. A sensor interpretation requires the mapping of sensor inputs to a particular concept. A group of sensor readings are firstly read into the application and the information contained in these readings is extracted and temporarily stored. The application then looks at all the sensor readings as a group and attempts to match them with a concept. A concept consists of two parts the first is a set of Fire Conditions (for a description of Fire Conditions see section 3.1.3) that are used in the interpretation. The second part consists of what action to take if the fire conditions are met and the concept is chosen. The actions are predetermined and are available to be looked up by the application. A diagram of this process is detailed in Figure 11 below.

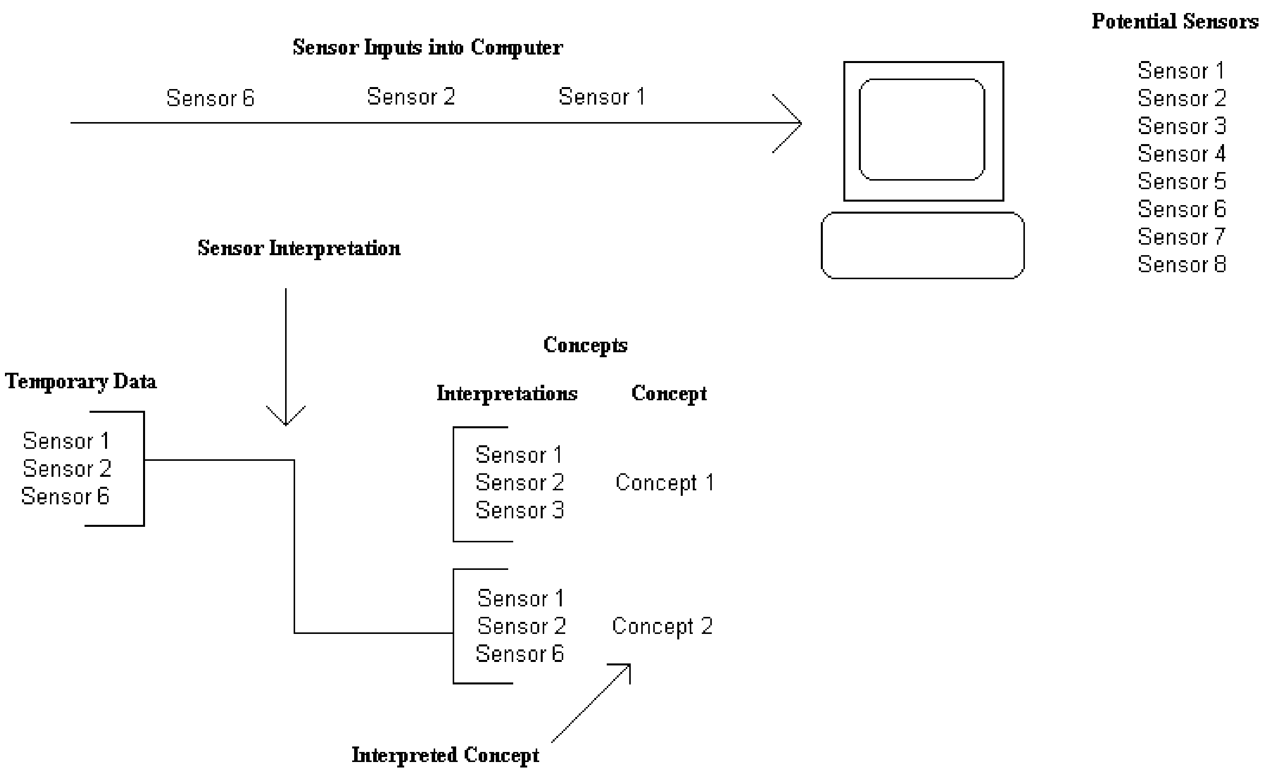


Figure 11: Sensor Interpretation

An example of where an interpretation is used could be a room that may be installed with a motion sensor, light sensor and pressure sensor on the floor in front of a projector. When the light sensor is reading a “light” room along with movement recorded in the motion sensor but no pressure on the pressure sensor an interpretation may be that the room is in a state of “pre lecture” and a music file may play. Alternatively the light sensor may have a “dark” room reading with or without motion and pressure indicated on the stage. This could be interpreted as a “lecture is underway” and PowerPoint is started, ready for the forthcoming lecture. The concept in these examples is a) “pre lecture” and b) “lecture is underway”.

3.1.2 Sensor Interpretation Ambiguity

When attempting to interpret sensor inputs from multiple sensors that lead to multiple interpretation concepts a unique interpretation that matches sensor inputs may not occur. An example of this featured in the diagram below is if a room has three sensors (one, two and three) and interpretation one is a combination of sensor one and two while interpretation two is a combination of sensor two and three. Which interpretation should be used if all of these are active?

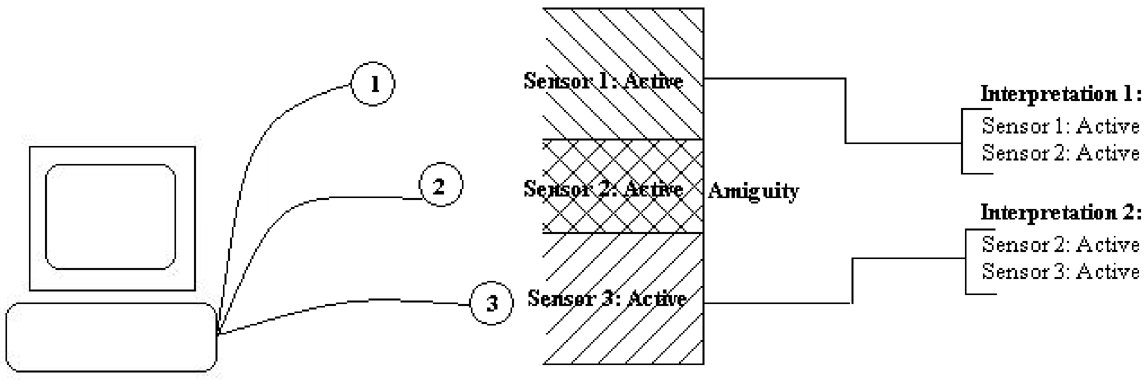


Figure 12: Sensor Interpretation Ambiguity

In the event of this Ambiguity a set of rules needs to be developed such that an interpretation can be selected as the correct option. Implementing “priorities” to interpretations would be a way to counteract these conflicts. By introducing a priority to interpretations the ambiguity in the above situation disappears as the higher priority interpretation is selected. The diagram below demonstrates that with the new priorities the ambiguity is gone and Interpretation 2 is selected.

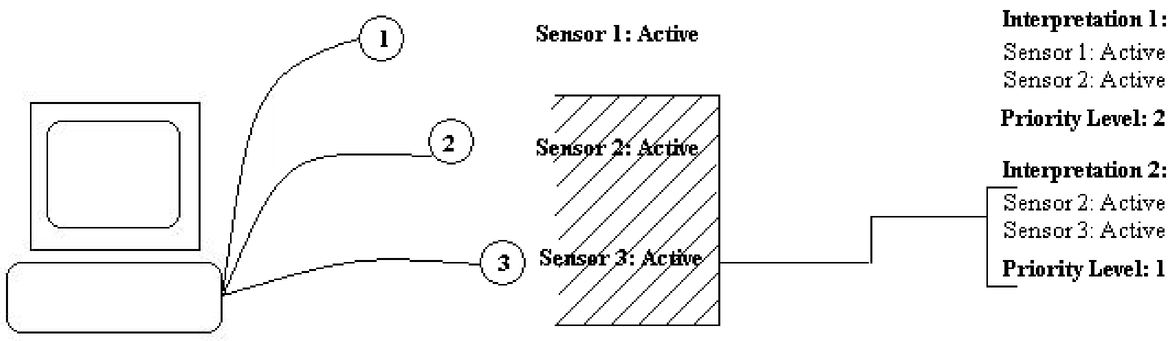


Figure 13: Prioritised Sensor Interpretations

For further analysis of sensor interpretation conflicts see chapter 4

3.1.3 Fire Conditions

As mentioned in the previous section a sensor interpretation requires the mapping of sensor inputs to a particular concept. In the HITSI Application Fire Conditions are used as part of this concept. A Fire Condition is essentially a single rule that can be compared to sensor inputs to give a true or false value. In Figure 13 above an example of a fire condition in interpretation 1 is “Sensor 1: Active”.

Every fire condition essentially consists of two parts. The first is the identification of which input this fire condition is to be compared against. In the previous example this would be “Sensor 1”. The second part of the fire condition is a rule that needs to be satisfied to give the fire condition a true value. The previous example requires the sensor to have an “active” state for the fire condition to be true.

Using these fire condition guidelines an interpretation can be attempted by mapping sensor inputs to fire conditions and evaluating how many fire conditions have been met.

3.1.4 Sensor Input Time Window

All human based sensor inputs cannot be performed and read at the exact same instant by a computer application, similarly humans would normally not interact with multiple sensors simultaneously, therefore an application using human input through multiple sensors needs to allow a short period of time for these human actions and sensor readings to be read and processed by the computer application. The following diagram shows an example of multiple sensor inputs occurring over a period of time.

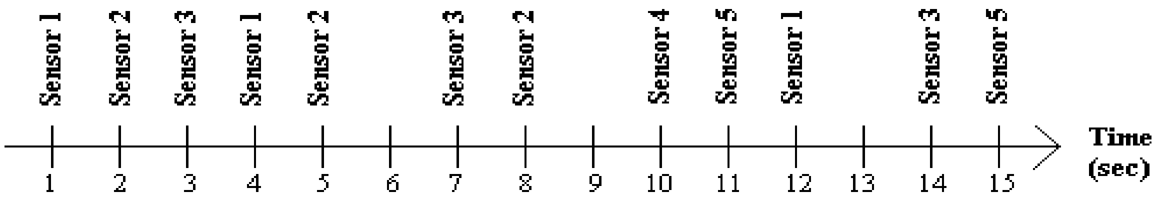


Figure 14: Multiple Sensor Inputs Over Time

It would be very difficult to attempt to interpret these sensor inputs as they arrived in this way because the amount of data available for interpretation grows constantly and interpretations could take place almost every second resulting in many undesired interpretations. The application needs a way in which to determine when an interpretation should be attempted.

The idea behind the time window is to allow the application a period in which to collect sensor input information before attempting an interpretation. The following diagram shows how the data presented in the previous example can be grouped into smaller more manageable time groups. These sections of time are the time windows.

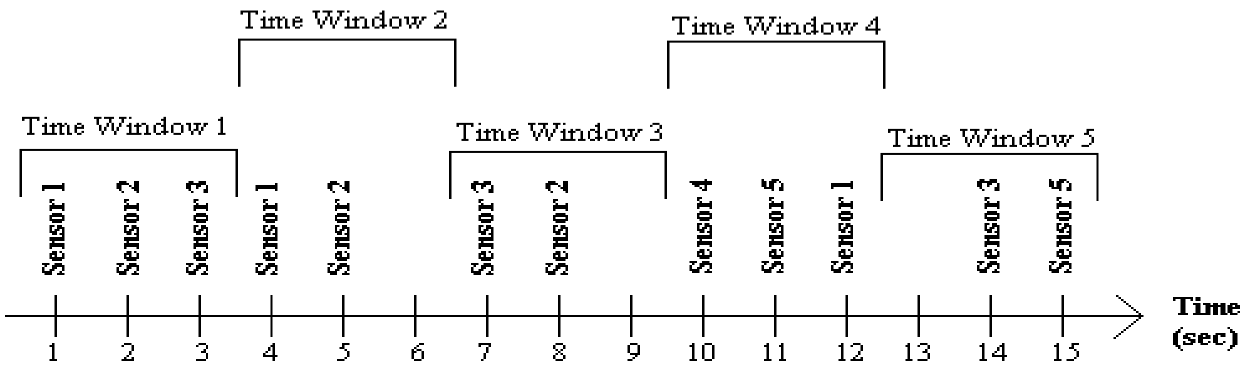


Figure 15: Multiple Sensor Inputs With Time Window

3.1.5 Sliding Time Window

I explored several possibilities in introducing a time window. The first is the use of a "sliding time window". This window would start when each sensor reading is read into the application and would monitor all other sensor readings until the time window collapses whereby an interpretation would take place. This method initially appealed to me as it only does processing and interpretations when sensor readings are recorded. However a serious issue arose when a model was explored.

Figure 16 demonstrates that the firing of sensor 1 and 3 could be used in two different interpretations. It is not possible to tell which interpretation the user has meant the sensors to fire for. At the instance when the first time window collapses information for future time windows cannot be accessed therefore it will interpret what it believes to be a correct interpretation of "Interpretation 1". The user however may have meant for the use of sensor 2 firing to be used in a previous interpretation and for sensors 1 and 3 firing to be interpreted as interpretation 2.

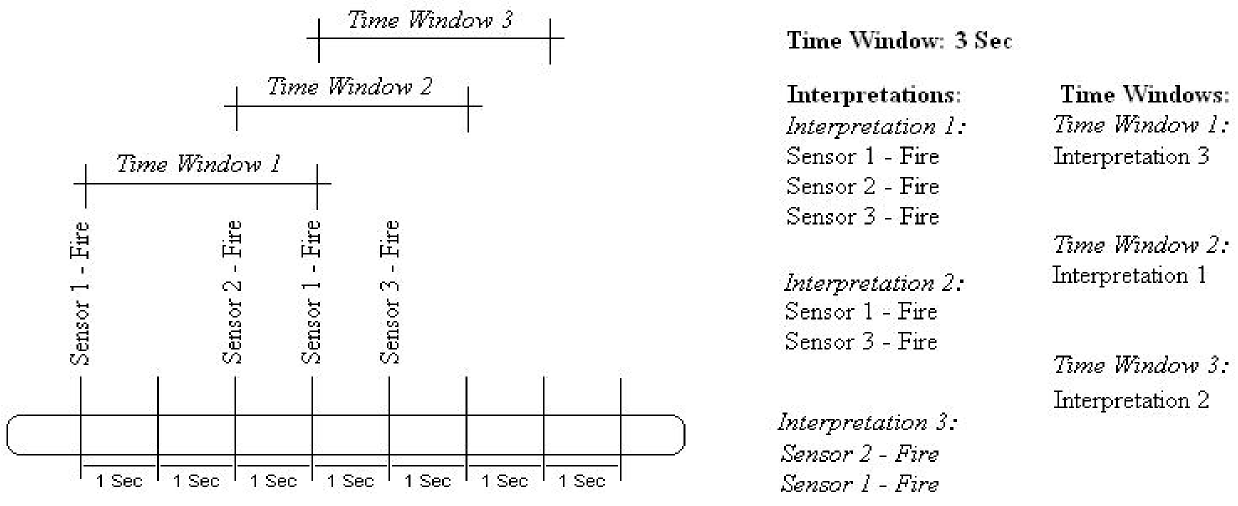


Figure 16: Sliding Time Window

To overcome the problems faced with the sliding time window I adopted a "static time window". A "static time window" collapses once every time window period and starts again only after the previous time window has collapsed. Figure 17 shows how a time window begins at the conclusion of the previous time window and extends for the predetermined time window duration. This solves one of the ambiguities that arise between interpretations as sensor inputs only ever occur in one distinct time window. Time windows are discussed further in section 4.3

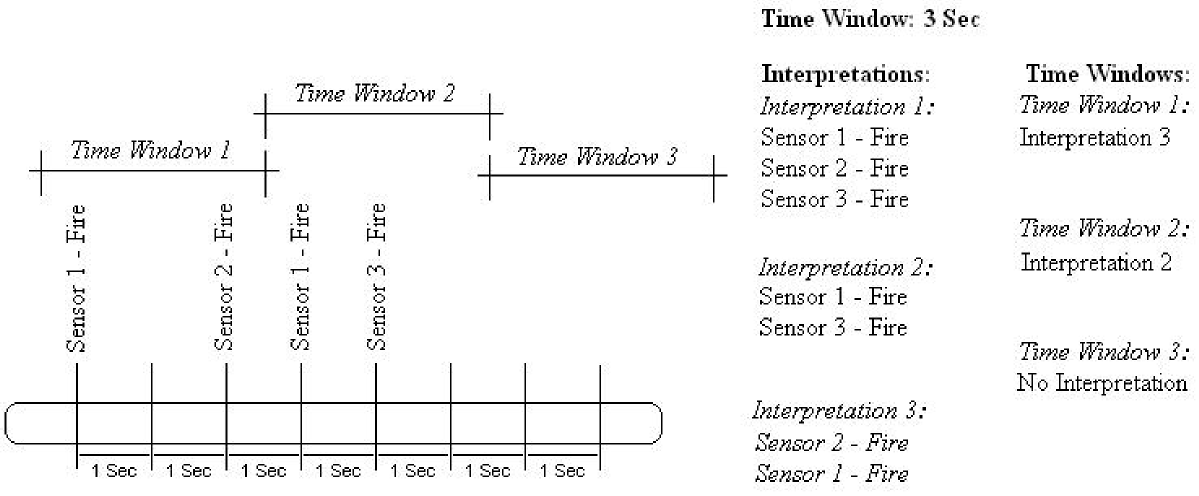


Figure 17: Static Time Window

3.1.5 Coupling

One of the requirements of the HITSI Application is the ability to control actions in a third party software application. When discussing two separate software applications relying on one another a decision on whether to use low coupling or high coupling soon follows.

Low coupling indicates a relationship between two applications that does not require either application to understand the others internal implementation, while high coupling indicates the reverse.

If a high coupling is used the HITSI Application could be severely restricted in what third party applications are available to the user. High coupling would require that at least some information from each third party app that is used to be included in the source code of the HITSI Application. This would create the issue that only those third party applications that have information in the HITSI Application source code could be used and any new third party application would require rewriting of the HITSI Application code and a recompiling of the project.

If low coupling is used the HITSI Application would not require recompiling for every new third party application used. Instead the HITSI Application would need to send messages to the third party application without having to know how the application uses them but only how to construct a message that it understands.

To meet the requirements of low coupling the HITSI Application could take advantage of windows system messages to send information to the third party application (See 4.3.3 for more information on system messages). These system messages are standard to all windows based applications and could therefore be understood and used by all third party windows based applications.

3.2 High Level Design

Using the design decisions listed in the previous chapter and the information obtained through the literature review a high level design can be created. The design in this chapter attempts to meet and further the requirements and aims listed in chapter one and provides solutions to issues that have arisen as a result of these.

3.2.1 Use Cases

The following use case outlines the potential tasks a user can undertake to achieve human interaction through sensor interpretation using the HITSI application. Tasks include creating new users, selecting preferences and using sensor inputs to command a third party application.

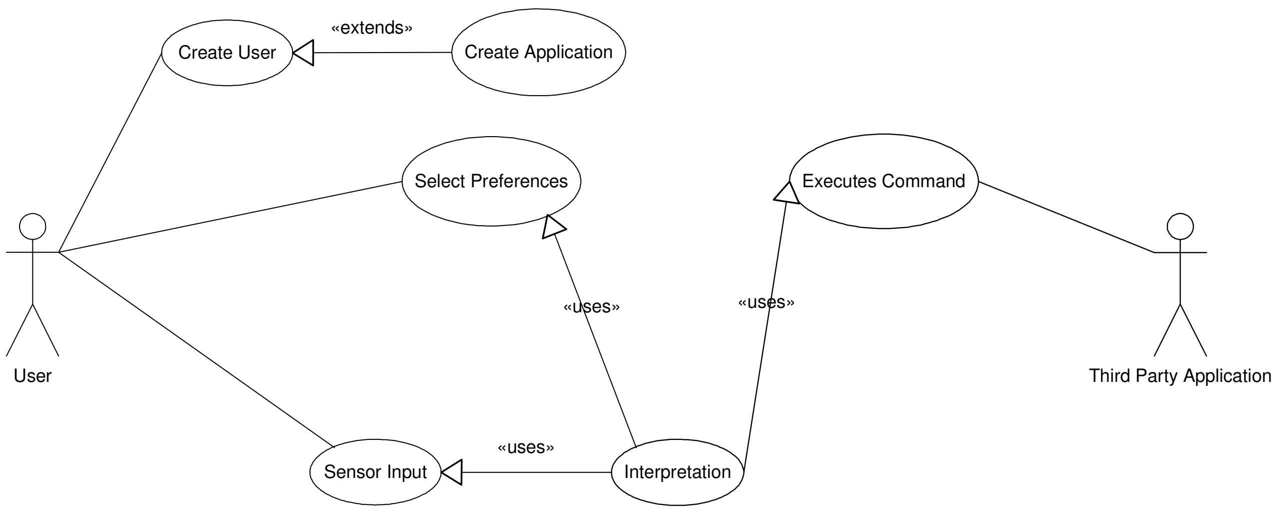


Figure 18: Use Case

Name: Create User

Description:

Adding a user to the Database

Preconditions:

- User does not already Exist

PostConditions:

- A new User will be added to the database

Basic Course of Action:

1. A new user wishes to create their own preferences in the database.
2. The user creates their own entry in the preferences database
3. The uniqueness of the user is checked (Alternate Course A: The user already exists)
4. The user saves the changes to the database
5. The use case ends

Alternate Course A:

4. The user is prompted that a preference file already exists
5. User exits database without saving

Name: Create Application

Description:

Adding application preferences to a designated user in the Database

Preconditions:

- User exists in database
- Application information does not exist for User in database

PostConditions:

- A new Application and preferences will be added to the database

Basic Course of Action:

1. A user wishes to create preference information about an application in the database.
2. The user finds their user information in the database
3. The application name is entered (Alternate Course A: Application Already exists)
4. Preference information is entered under the new application in the database.
5. The user saves the changes to the database
6. The use case ends

Alternate Course A:

4. The user is prompted that the application name is already contains preference information.
5. User exits database without saving

Name: Select Preferences

Description:

A user selects user name, application and file name for execution.

Preconditions:

- User exists in database
- Application exist for user in database

PostConditions:

- Selected preferences will be chosen and ready for application execution

Basic Course of Action:

1. User selects 'username' from a list on application screen
2. User selects an application from screen (Only applications that exist for the selected user are available for selection)
3. User enters file path and file name to be used by third party application.
4. User presses a "Start Sensor Monitoring" button
5. System stores selected data for use in an "interpretation" use case
6. The use case ends

Name: Interpretation

Description:

After a time period expires the stored sensor input information is analysed to decide upon an action.

Preconditions:

- Sensor Inputs have been recorded
- Concept information for interpretation is available

PostConditions:

- A command is issued to the third party application

Basic Course of Action:

1. A Time period expires
2. Each sensor input is used and compared against the concept information in the form of an interpretation
3. An interpretation is chosen as the best match(Alternate Course A: No interpretation meets provides a suitable match)
4. The action associated with the interpretation concept is extracted and passed to the third party application for execution
5. All sensor inputs are cleared ready for the next time period to begin
6. The Use case ends

Alternate Course A:

4. All sensor inputs are cleared ready for the next time period to begin

The Use case ends

Name: Sensor Input

Description:

User performs an action that that is picked up by a sensor and stored by the application for later use.

Preconditions:

- User has selected preferences and pressed "Start Sensor Monitoring Button"

PostConditions:

- The sensor input is stored for later use in an interpretation

Basic Course of Action:

1. User performs an action that triggers a sensor reading to be passed to the application
2. Application detects sensor reading and records sensor information for later use
3. The Use Case Ends

Name: Executes Command

Description:

A third party application has a command executed from a remote source

Preconditions:

- Third Party application is currently running

PostConditions:

- The third party application executes a command

Basic Course of Action:

1. The third part application receives an external command via the HITSI application
2. The third party application process the command

1. The Use case ends

3.2.2 Block Diagram

The block diagram in Figure 19 shows the way in which the information flows in the HITSI application design. Sensors take readings and forward these onto the Phidgets Interface Control Board. From here sensor inputs are passed into the HITSI application. The HITSI application is able to monitor sensor inputs and store them for future interpretation. By comparing sensor inputs to stored interpretation definitions an interpretation is chosen and a system message is invoked to the third party app.

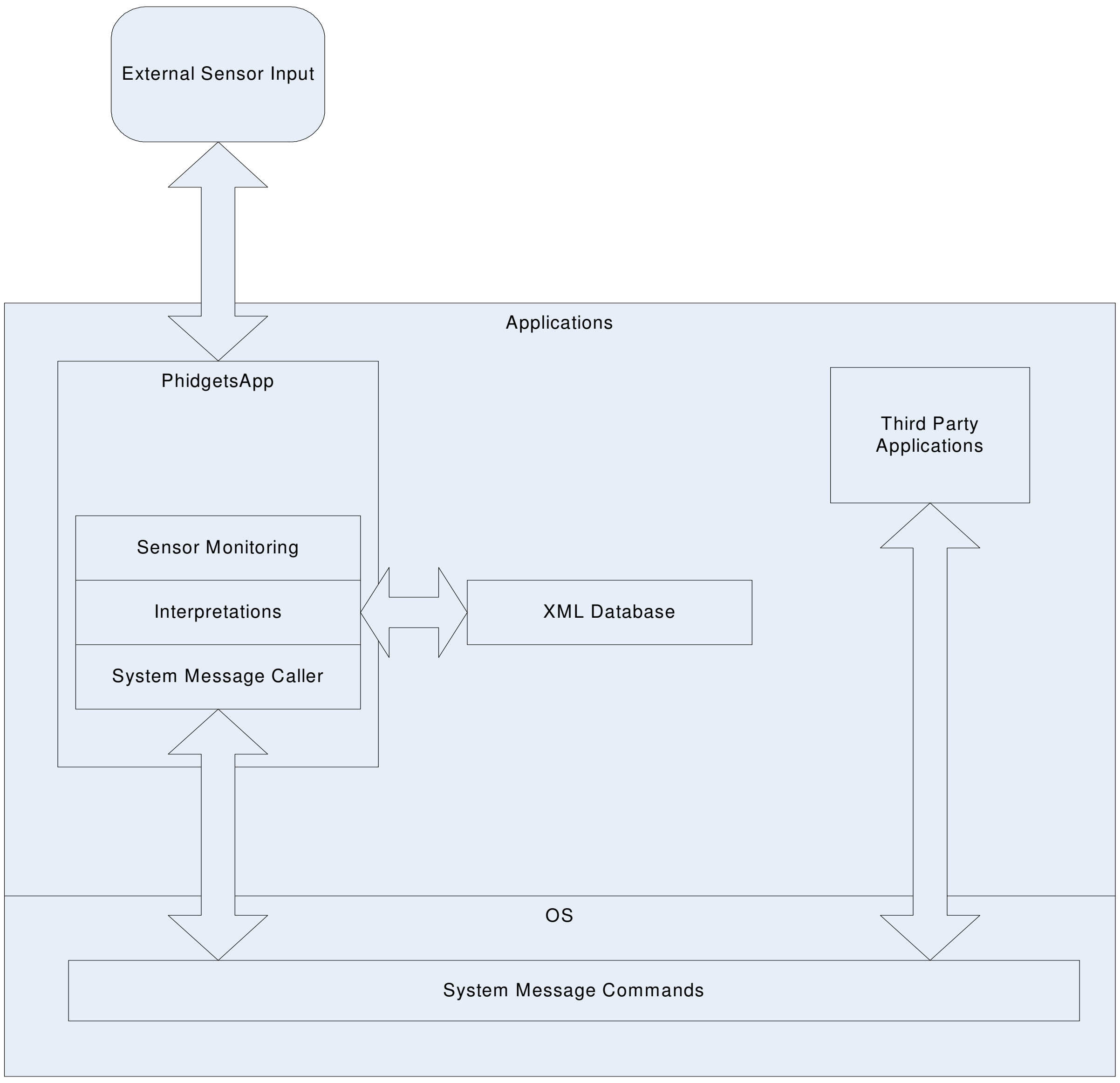


Figure 19: Block Diagram

3.2.3 Flow Chart

The flow chart in Figure 20 steps through the possible states that the HITSI application experiences during operation.

Initially a user must select their user name. User names are taken from the external database and are available for selection in the HITSI Application. Only configuration data relating to that user will be available from this point on. This data consists of preference information relating to third party application information and interpretation information.

Once a user has selected their username they must select which third party application they wish to control. Only options that are stored within the database under the selected user are available.

A user must then select the file in which they will control through the third party application. The file name is used to identify the correct window of the third party application in the event that two copies of the same application are open at the same time.

As soon as a user has identified the Username, Application and file name they are ready to begin monitoring sensor inputs ready for interpretations.

The next action the HITSI application undertakes is a wait until a sensor input is detected. Upon detection of a sensor input a check must be made to ascertain whether the timer has been initiated or not. If the timer has been activated the sensor input is stored and the HITSI application continues to wait for the next sensor input.

If the timer has not been activated the HITSI application continues the same as if the timer had not been activated except that the timer is activated. While the timer is counting sensor inputs continue to be stored.

When a predetermined time has elapsed since the sensor input that triggered the timer was received the timer stops. Immediately after the timer stops all sensor readings up to that point are grouped together and an interpretation takes place. After the interpretation the resulting action is executed in the third party application. The system then returns to waiting for the next sensor input.

NOTE: In the case where the timer has started then been stopped but before the third party application action has been executed any sensor inputs will be recorded under the new timer because a check will be made to see if the timer is active (which it is not) and the sensor input will be stored and the timer started again.

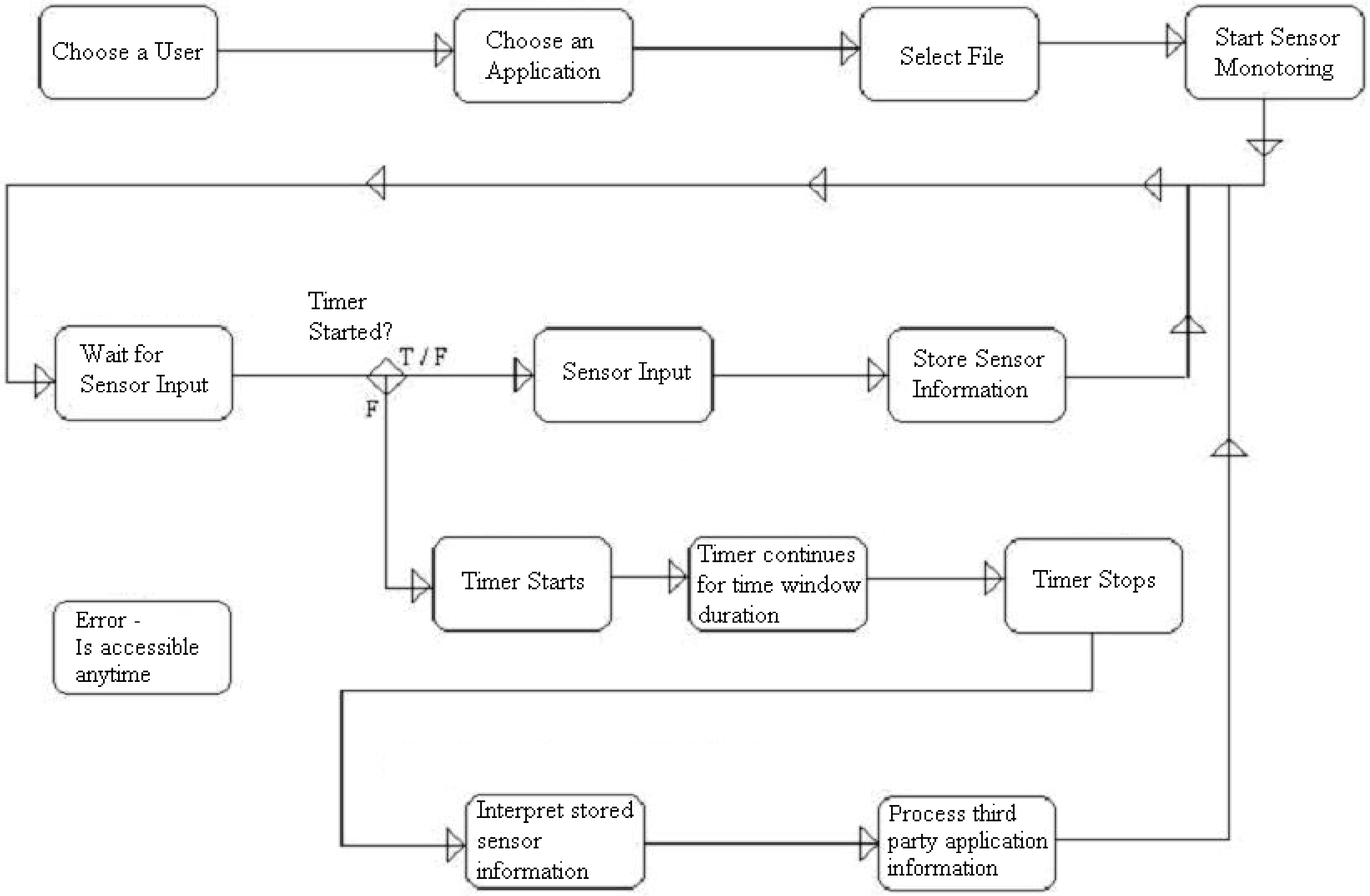


Figure 20: State Diagram

3.3 Summary

Chapter 3 approached the high level design of the HITSI Application. One of the major design issues that arose was handling multiple sensor inputs. Concepts including Sensor Interpretations, Fire Conditions and Time Windows were introduces to solve these issues. The chapter also included some formal design for the HITSI application in the form of UML diagrams. Use Cases, a Block Diagram and a Flow Chart were used to outline the approach the HITSI application used to meet the design aims and issues covered in the previous chapters.

Chapter 4 HITSI Application Implementation

This chapter outlines the details of how the HITSI application was implemented following the design outlined in chapter 3. This chapter describes the technologies that were used in creating the HITSI Application prototype, several UML diagrams to explain the finer details of the inner workings of the HITSI Application and provides further analysis of Time Windows and Sensor Interpretations, as they have been implemented in the prototype.

4.1 Technologies Used

The following chapter lists the technologies used in developing the HITSI application prototype. Technologies include the development platform, Database decisions the Sensor Package used for sensor input and other applications used to develop the prototype. The decisions leading to these choices are also explored.

4.1.1 Microsoft Visual Studio .NET 2005 (Visual C#)

The decision to use Microsoft Visual Studio .NET 2005 (Visual C#) as the development platform was influenced by the online support available, the compatibility of the API's provided with the Phidgets Interface Toolkits, the ease and support for interoperability with XML and the availability of the development platform.

4.1.2 XML

XML was chosen as the format to store database information. XML APIs are included within the Microsoft Visual Studio .NET 2005 (Visual C#) libraries and provided all functionality required to successfully build the HITSI Application prototype. XML allows for users to modify stored information without the addition of extra software and also provides information in a structured and easy to read format.

4.1.3 Spy++

Spy++ was used to obtain the system messages used in the prototype and could also be used to obtain future system messages. Spy++ is a windows application released within the Visual Studio 2005 .NET tools and is used to obtain a list of the systems processes, threads, windows and window messages [SPY++ Help file].

4.1.4 Sensors Used

For the prototype implementation of HITSI Application I used a Phidgets Interface Kit (PIK) as can be purchased from http://www.phidgets.com. The PIK (pictured below) is a lightweight package from Phidgets USA that enables the use and control of sensors from a PC. The PIK consists of 8 analogue inputs, 8 digital inputs, 8 digital outputs and a 2-port USB hub. The Phidgets Interface Kit connects to a PC through a USB connection.

The analogue inputs on the PIK can be used to measure continuous sensor readings such as those in measuring pressure, motion, light, etc. The digital inputs can be used to determine the state of buttons, limit switches, relays, etc

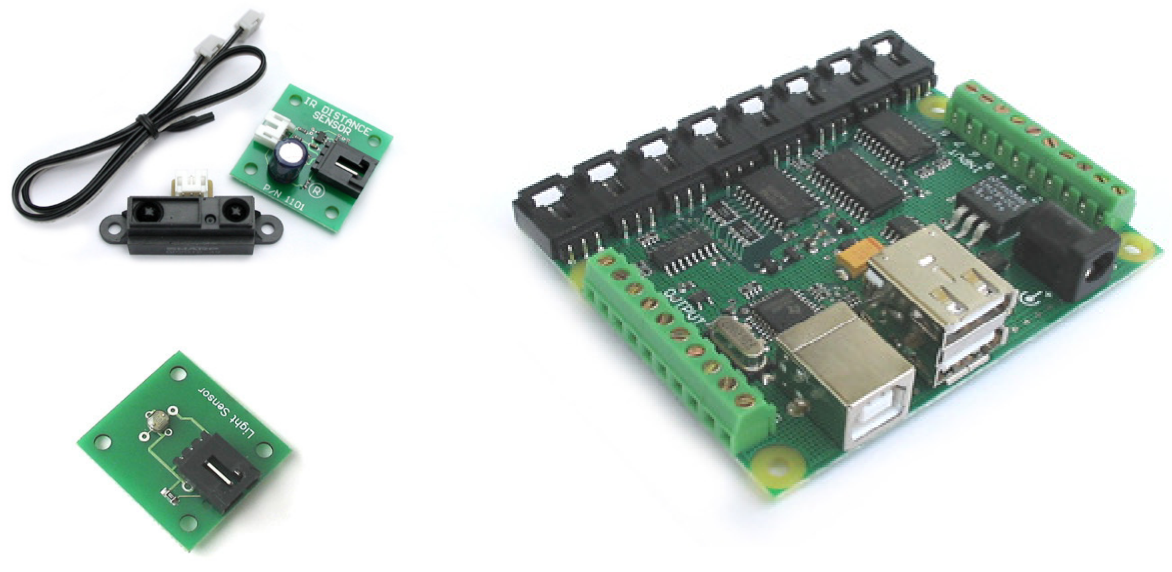
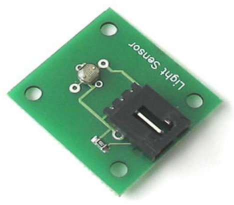


Figure 21: Phidgets Interface Kit

The PIK also comes with a software library that takes only moments to install and use. The PIK can be controlled from Windows, Mac OS and Linux. For software development high level programming languages such as Visual Basic, C++, C#, .NET, labVIEW, etc are all supported.

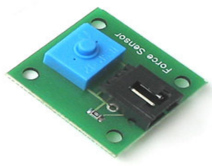
The following is a list of Phidgets Sensors that can be attached through either the digital or analogue ports of the PIK(note that the following is not an extensive list, only sensors that were used in the trial of the HITSI Application prototype)



Light Sensor:

Attached through an Analogue Port on PIK

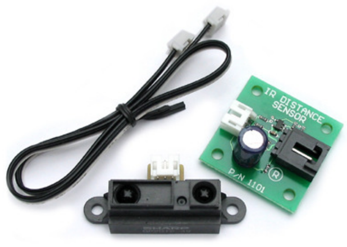
A standard CdS photocell resistance varies with light



Force Sensor:

Attached through an Analogue Port on PIK

Measures Force however is not accurate enough to used as a weight measuring device



IR Distance Sensor:

Attached through an Analogue Port on PIK

Measures distances from 10cm to 70cm

Motion Sensor:

Attached through an Analogue Port on PIK

Infrared motion sensor concentrating on a 15-degree cone from front of sensor.

4.2 Low Level Design

This section makes use of several UML diagrams including Class diagram, Sequence Diagrams and Component diagrams to explain the inner workings of the HITSI Application. The internal structure of the Phidgets Application is explained as well as the way in which the HITSI application processes sensor inputs all the way from sensor monitoring through to interpretations and executing a command in the third party application.

4.2.1 Class Diagram

The class diagram in Figure 22 Illustrates the classes used in the development of the HITSI application including methods, attributes and the relationships between these classes. The UI classes (UI Main and UI Sensor Interpreter) are actually representations of the two screens used in the HITSI Application. As the HITSI application aims to command another application through the use of sensors the user only views these screens during the start of the HITSI application.

The XMLData DB and Error DB classes are representations of the two XML files used to store configuration options and error details. For a closer look at these databases refer to section 4.2.4

The Action_Interpretor class provides access to all the information relating to sensor inputs and interpretations. Sensor Inputs are stored temporarily in the ActionClass during a timewindow and are removed after being involved in an interpretation.

All possible interpretations (see section 3.1.1 for definition of interpretation) are stored in the Interpretation class and each fire condition (see Section 3.1.3 for definition of fire condition) for these interpretations is stored in the Fire_Cond Class.

After an interpretation is selected the Third Party class is used to interact with a third party application.

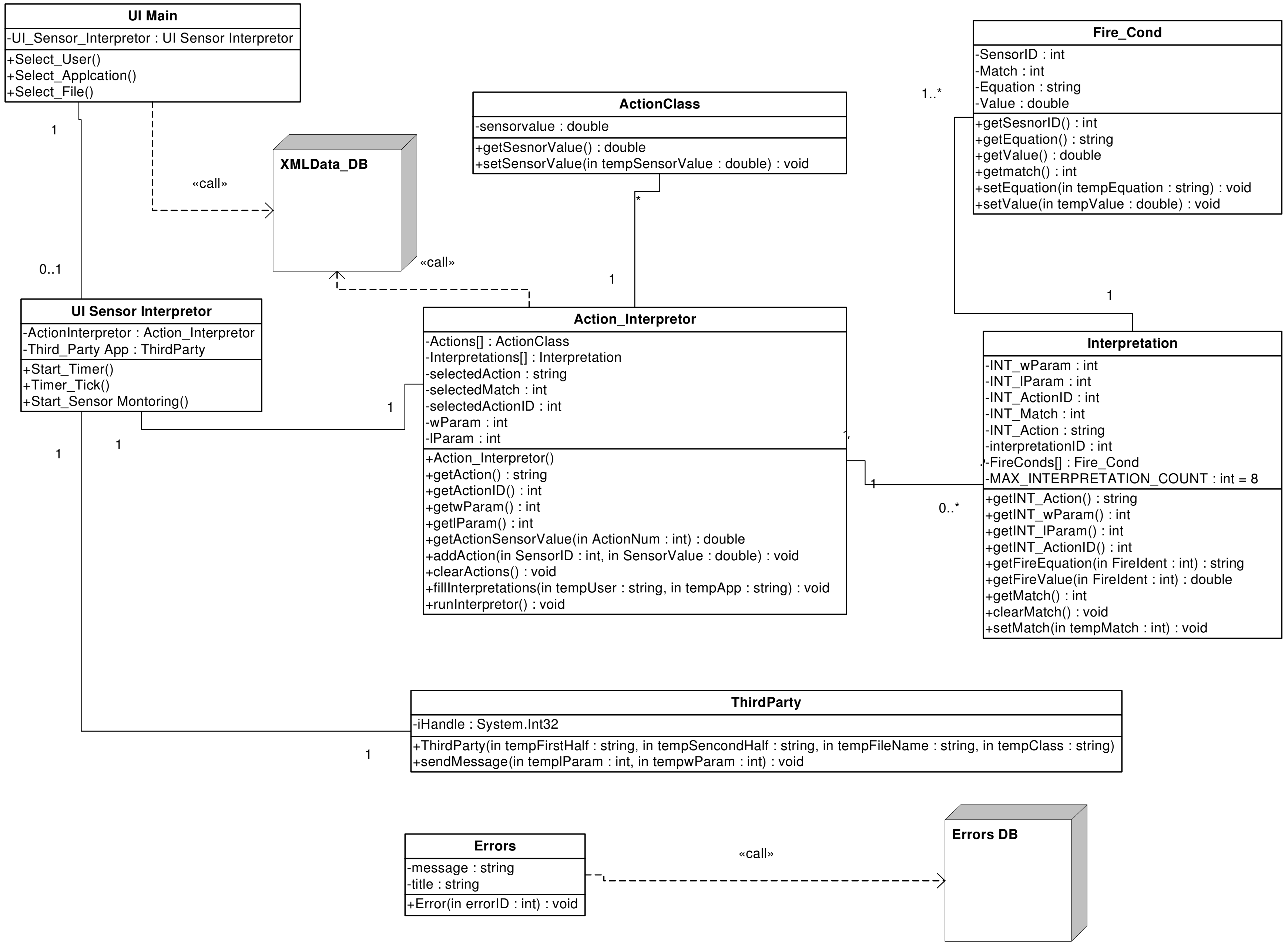


Figure 22: HITSI App Class Diagram (and how the DB is used)

4.2.2 Sequence Diagrams

The following three sequence diagrams outline how a user begins the application by selecting user details and how the HITSI application processes time windows and interpretations.

4.2.2.1 Starting Application

When the HITSI application is started all Users usernames are read from the XMLData DB and presented to the user in a combo box. After a user selects there username the HITSI application queries the XMLData DB to find all application entries for that user. These Application names are displayed to the user in a separate combo box. A user must then enter the directory and filename of the file in which the third party application will be using. (For more details on the use of usernames, application names and filenames, refer to section 4.2.4)

After a user has successfully selected and entered the previously listed details they are able to press the start program button on the Main Screen. This button starts the HITSI application sensor monitoring and the sequence diagram in Figure 24 continues on directly where Figure 23 finishes.

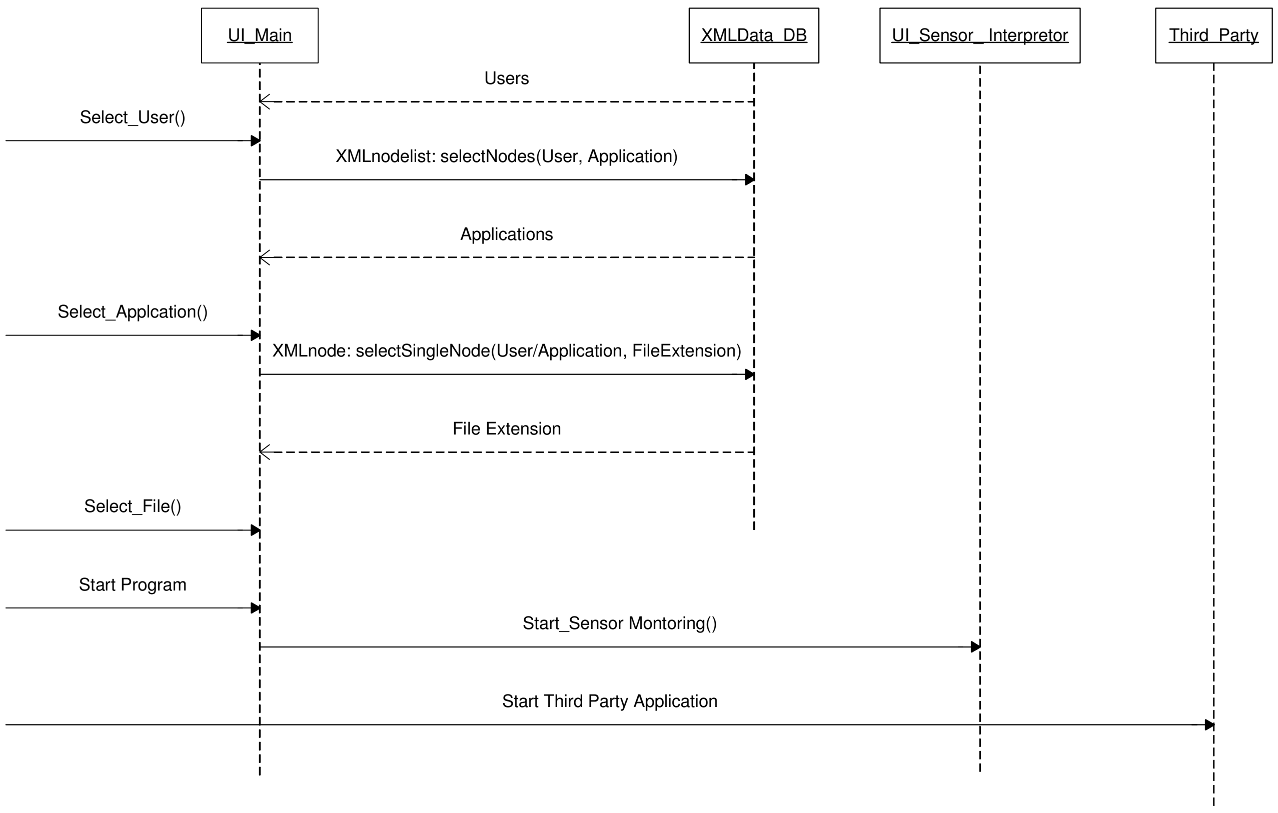


Figure 23: Starting Application Sequence Diagram

4.2.2.2 Timer / Interpretation Sequence Diagram

When one of the sensors used by the HITSI application detects a change in value (Movement in a motion sensor) the ifkit_SensorChange function is called. When this event occurs a timer is started within the HITSI Application. For a description on the timer please refer below to Shaded Box 2: Timer.

The next action the HITSI application takes is to pass the data obtained from the ifkit_sensorChange call through to the action interpretor. The Action_Interpretor then stores this information in the Action class, which is called using the setSensorValue function.

After a period of time passes that is referred to as a time window (as described in section 3.1.4) the Timer_Tick function fires. Within the timer tick function the runInterpretor functions is called. This starts the process of interpretation. After the Action_Interpretor receives the runInterpretor function call it makes calls to the ActionClass and Interpretation classes in order to obtain the information required to conduct the interpretation (For a more detailed explanation on the interpretation process refer to section 4.3.2).

Once the interpretation has taken place and a match selected the HITSI Application retrieves the information required to command the third party application from the interpretation class. After this information is obtained a call to the ThirdParty class using the sendmessage function is issued. The information obtained in this message is used by the ThirdParty Class to identify the third party application and issue the command contained within the sent information. (For more information on commanding the third party application refer to section 4.3.3)

Shaded Box 1: Sensor Inputs

The methods contained within this box are the result of sensor inputs being received by the HITSI Application. These sensor inputs and thus methods can be called randomly through the entire process including during interpretations.

Shaded Box 2: Timer

The Timer operates independently from the sensor inputs and thus a Timer_Tick can occur during a sensor input. The period between the Start_Timer and Timer_Tick methods is defined by the time window size. For more information about time windows please refer to section 4.6.

Shaded Box 3: Interpretation

The methods contained within this box are used for the purpose of making an interpretation. The number of calls to the ActionClass is limited to eight as this is the amount of sensors that are available in the HITSI Application. The number of calls to the Interpretation and Fire_cond Classes depends upon how many interpretations and fire conditions are available for this application and user. After each setMatch function is called the match value (as outlined in section 4.3.2) is also compared to the stored Match information. If the Match value is higher then the stored "selected interpretation" value then the new interpretation information is stored as the new "selected interpretation". For a view of the values being exchanged refer to Figure 25.

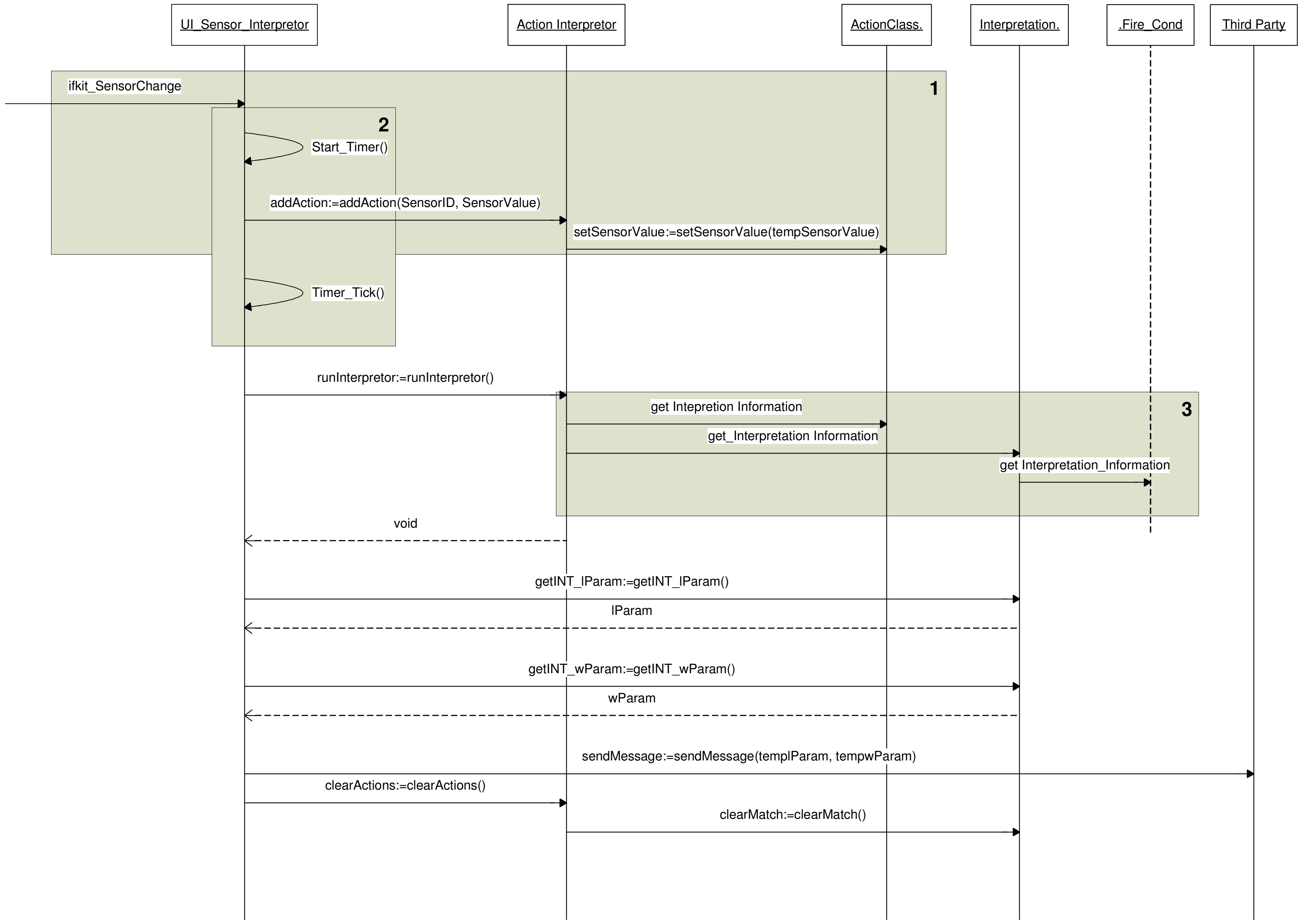

UI_Sensor_Interpretor
Action Interpretor
ActionClass.
Interpretation.
.Fire_Cond
Third Party
1
ifkit_SensorChange
2
Start_Timer()
addAction:=addAction(SensorID, SensorValue)
setSensorValue:=setSensorValue(tempSensorValue)
Timer_Tick()
runInterpretor:=runInterpretor()
3
get Intepretion Information
get_Interpretation Information
get Interpretation_Information
void
getINT_lParam:=getINT_lParam()
lParam
getINT_wParam:=getINT_wParam()
wParam
sendMessage:=sendMessage(templParam, tempwParam)
clearActions:=clearActions()
clearMatch:=clearMatch()

Figure 24: Timer / Interpretation Sequence Diagram

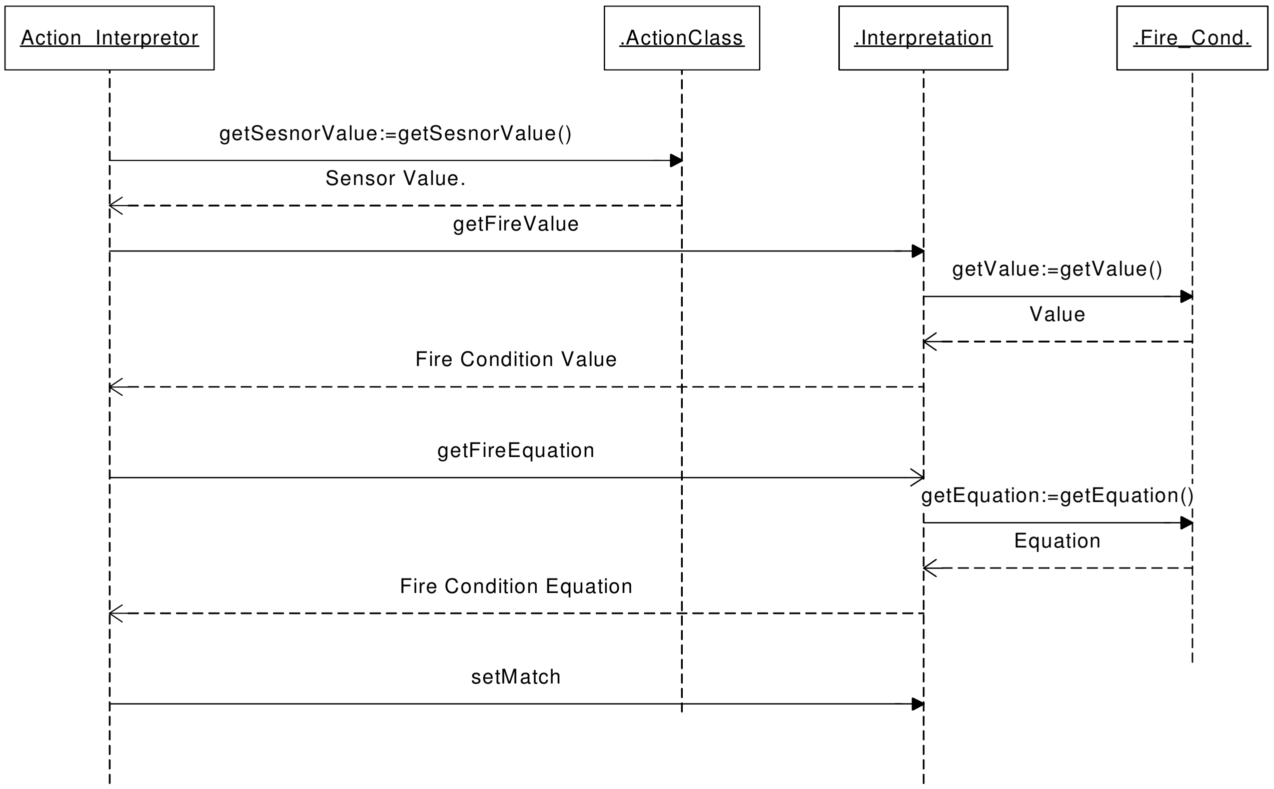


Figure 25: Interpretation Information Sequence Diagram

4.2.3 Component Diagram

Figure 26 shows the components in the HITSI application and there relationships. Each component in the diagram contains one or more classes as displayed in the class diagram in section 4.2.1

Component: DB
Classes: XMLData DB, Errors DB
Interactions: The DB is queried by the interpreter and Error components and provides data that was contained within the database.

Component: Interpretor
Classes: Action Action_Interpretor, ActionClass, Interpretation, Fire_Cond
Interactions: The interpretor component uses information provided by the DB component to perform interpretations when called to do so by the UI component.

Component: UI

Classes: UI_Sensor_Interpretor, UI_Main

Interactions: The UI component consist of the User interface screens that the user interacts with. All sensor inputs come through this component and control interpretations and calls to third party applications.

Component: Error

Classes: Errors

Interactions: The Error component is accessed by all components and provides information to the user when an error has occurred in any of the other components.

Component: Third Party Application

Classes: ThirdParty

Interactions: The third party application component takes instructions from the UI component and sends messages to an actual third party application

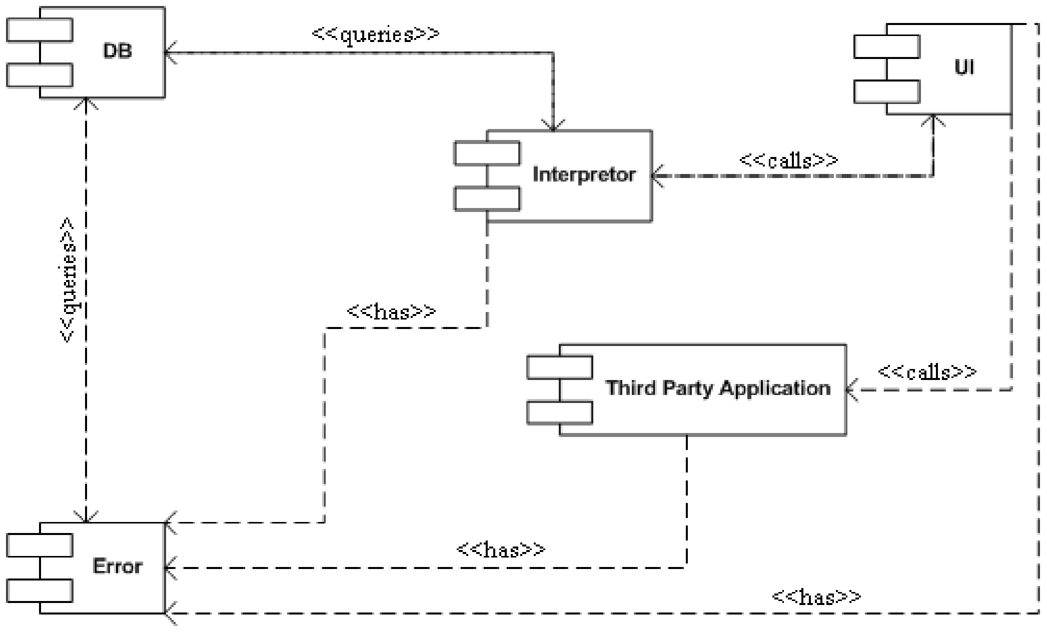


Figure 26: Component Diagram

4.2.4 DB Design

The database design for the HITSI application is required to be comprehensive in the amount of information that can be stored and also easy to read and modify so that users are able to easily customize the information contained within. The following section gives an example of the XMLData.xml file as well as a description on when and how data is extracted as well as descriptions on each tag in the file. The DB can be modifed and/or created by a user through the use of a text editor.

4.2.4.1 Errors.xml Example

The following is an example Errors.xml file. The example only contains three errors and there corresponding messages.

```
<Errors>
        <Error>
                 <ErrorID>1</ErrorID>
                 <ErrorMessage>Please Select a User</ErrorMessage>
                <ErrorTitle>Insufficient Data</ErrorTitle>
         </Error>
         <Error>
                 <ErrorID>2</ErrorID>
                 <ErrorMessage>Please Select a User and Application</ErrorMessage>
                <ErrorTitle>Insufficient Data</ErrorTitle>
          </Error>
          <Error>
                  <ErrorID>3</ErrorID>
                  <ErrorMessage>Could not find XMLdata.xml</ErrorMessage>
                  <ErrorTitle>Missing File</ErrorTitle>
          </Error>
</Errors>
```

Certain errors that are found to occur within the HITSI application have an entry within the Errors.xml file. When these errors occur the application uses the error id provided and looks up the Error from within the XML file. Once the data has been extracted the HITSI application creates a message box using the information obtained and displays the information for the user.

4.2.4.2 XMLData.xml Example

The following is an example XMLData.xml file. This example only contains one user, one application and one action.

```xml
<Config>
 <User>
  <UserName>Chris</UserName>
  <WaitTime>1000</WaitTime>
  <App>
   <AppName>PowerPoint</AppName>
   <FileExtension>ppt</FileExtension>
   <ClassName>screenClass</ClassName>
   <FirstHalfWindowName>PowerPoint Slide Show - [</FirstHalfWindowName>
   <LastHalfWindowName>]</LastHalfWindowName>
   <FileName>true</FileName>
   <Action>
    <ActionName>Next Slide</ActionName>
    <wParam>00010189</wParam>
    <lParam>0x00000000</lParam>
    <FireOnce>True</FireOnce>
    <ActionID>1</ActionID>
    <FireCond>
     <SensorID>0</SensorID>
     <Equation>Greater Then</Equation>
     <Value>200</Value>
    </FireCond>
    <FireCond>
     <SensorID>1</SensorID>
     <Equation>Greater Then</Equation>
     <Value>500</Value>
    </FireCond>
    <FireCond>
     <SensorID>2</SensorID>
     <Equation>Less Then</Equation>
     <Value>100</Value>
    </FireCond>
    <FireCond>
     <SensorID>3</SensorID>
     <Equation>Greater Then</Equation>
     <Value>300</Value>
    </FireCond>
   </Action>
  </App>
 </User>
</Config>
```

The example above of the XMLData.xml file is explained in English below:

- One User with a username of Chris and a time window of one second
- One application called "PowerPoint" with file extensions ".ppt" is available
- The PowerPoint third party application is identified by the Classname "screenClass" and the window name "PowerPoint Slide Show – [ExampleFile.ppt]"
- One action is identified "Next Slide" and can be called in the third party application using values of "00010189" and "00000000" for wParam and lParam respectively.
- The Fire conditions for the Next Slide action are:

  Sensor 0 must be greater then 200

  Sensor 1 must be greater then 500

  Sensor 2 must be less then 100

  Sensor 3 must be greater then 300

The data stored in the above file is read into the HITSI application at different times for use in the application. The "UserName", "AppName" and "FileExtension" fields are all used by the HITSI application on the main screen when a user is selecting what data they will be using to drive there third party application.

Information is extracted from the database using the "SelectNodes" or "SelectSingleNode" functions included in the System.Xml library available to Microsoft Visual Studio .NET 2005 C#. A standard "SelectNodes" function call (Taken from HITSI source code) looks as follows:

```
        XmlNodeList users;

        XmlElement root = docXMLFile.DocumentElement;
        users = root.SelectNodes("/Config/User/UserName");
```

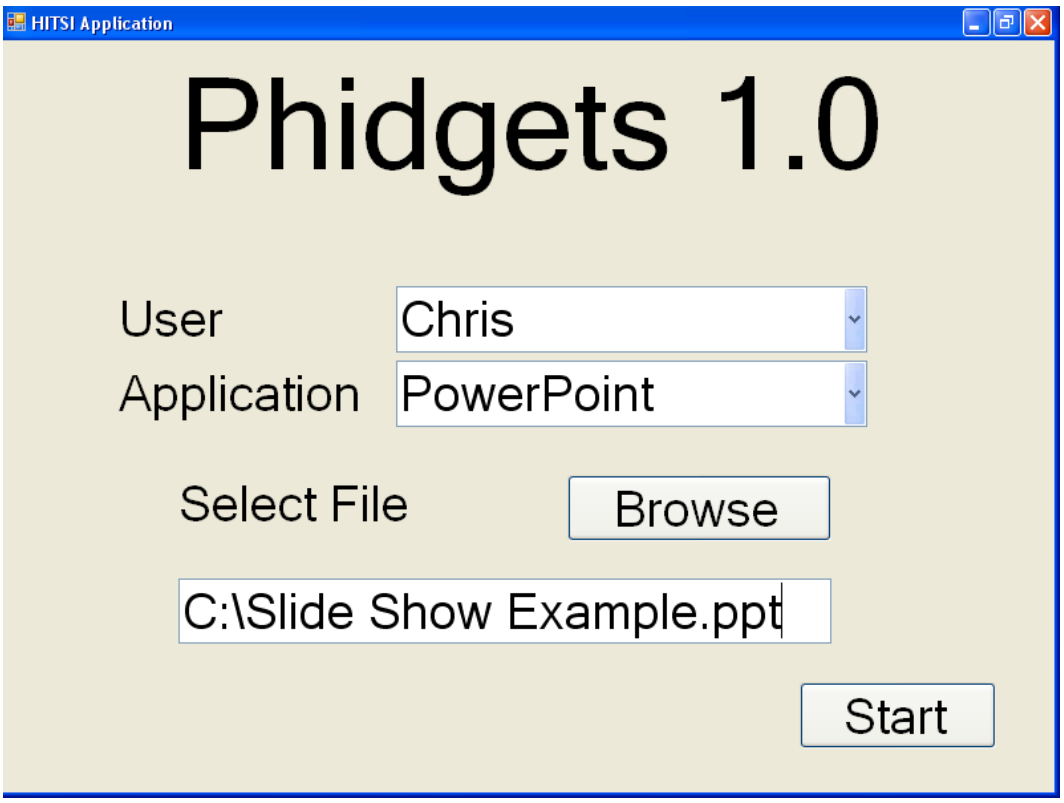
Assuming: docXMLFile points to XMLData.xml.

This piece of code looks in the XMLData.xml file for any nodes that exist that have a node path of "/Config/User/UserName" and returns all the nodes that meet this criteria. For example if the XMLData.xml file looked as follows (A simplified version of the example at the start of the section):

```
<Config>
        <User>
                <UserName>Chris</UserName>
        </User>
        <User>
                <UserName>Seng</Seng>
        </User>
</Config>
```

The XmlNodeList "users" would contain two nodes. The first would be "Chris" and the second "Seng".

The data relating to the application (ie <ClassName>, <FirstHalfWindowName>, <LastHalfWindowName> and <FileName>) is all used after valid selections of "UserName" and "AppName" and a valid file has been selected from the main screen and the "Start" button has been pressed. Figure 27 shows a screenshot of where the information is used



User Chris

Application PowerPoint

C:\Slide Show Example.ppt

Figure 27: Main Screen - Screen Shot

Data contained within <Action> and <FireCond> nodes are read from the XML file soon after the "Start" button has been pushed. This data is stored in the "interpretation" class for use when an attempt is made for an interpretation. The data contained within each <FireCond> is used to check against the sensor values obtained by the HITSI application to decide upon an interpretation. Once an interpretation is chosen (If sensor inputs do not meet interpretation rules no interpretation is selected. see Chapter 4) the data that was within the <Action> node is used to identify and command the third party app. For more information relating to controlling the third party app see section 4.3.3

4.2.4.3 Errors.xml

Errors.xml is used by HITSI Application to lookup errors in the XML database and provides the user with the resulting information.

Example XML:

```
<ErrorID>3</ErrorID>
<ErrorMessage>Could not find XMLdata.xml</ErrorMessage>
<ErrorTitle>Missing File</ErrorTitle>
```

XML Description:

<ErrorID>

Description: This field is used to uniquely identify each Error by an ID number

Example: <ErrorID>1</ErrorID>

<ErrorMessage>

Description: This field contains the error message that is presented to the user when the error occurs

Example: <ErrorMessage>Could not find XMLdata.xml</ErrorMessage>

<ErrorTitle>

Description: This Field contains the text that will displayed in the title of the message box when an error occurs

Example: <ErrorTitle>Missing File</ErrorTitle>

4.2.4.4 XMLData.xml

XMLData.xml contains information about user preferences, Application information and Interpretation Information.

XML Description:

<UserName>

Description: This field contains the name of the user whose preference information is contained in the sub fields.

Example: <UserName>Chris</UserName>

<WaitTime>

Description: This field is used to store the size of the time window to be used within the HITSI Application. (note: a value of 1000 is equivalent to 1 second, also values should only be of an integer type)

Example: <WaitTime>1000</WaitTime>

<App>

Description: The children fields of this field contain all of the information relating to one application that is used by the user.

Example:

```
<App>
        <AppName>PowerPoint</AppName>
        …
</App>
```

<AppName>

Description: This field contains the name of the application whose preference information is contained in the sub fields.

Example: <AppName>PowerPoint</Appname>

<FileExtension>

Description: This field contains the file extension of the files used by the application. This field can only contain one file extension and this file extension is used to filter files when browsing in the HITSI Application.

Example: <FileExtension>ppt</FileExtension>

<ClassName>

Description: This field contains the name of the window class the application uses during runtime. This information can be obtained through the use of spy++ (refer to section 4.3.3)

Example: <ClassName>screenClass</ClassName>

<FirstHalfWindowName>, <LastHalfWindowName> and <FileName>

Description: These fields are used as identification of the applications running window caption. Some applications use the name of the file in identifying it at runtime while others do not. For applications such as Microsoft PowerPoint that do use the file name FirstHalfWindowName and LastHalfWindowName need to be used to identify the text used before and after the filename when identifying the applications running window caption. Microsoft Media Player does not use the filename and all identifying text can be placed in FirstHalfWindowName. If the filename is used then <FileName> should be "true" otherwise it should be "false".

Example 1:

```
<FirstHalfWindowName>PowerPoint Slide Show - </FirstHalfWindowName>
<LastHalfWindowName>.ppt]</LastHalfWindowName>
<FileName>true</FileName>
```

The resulting window caption would be "PowerPoint Slide Show – Test.ppt" for Test.ppt

Example 2:

```
<FirstHalfWindowName>Windows Media Player </FirstHalfWindowName>
<LastHalfWindowName></LastHalfWindowName>
<FileName>false</FileName>
```

The resulting window caption would be "Windows Media Player" for Test.wav

<Action>

Description: The children fields of this field contain all of the information relating to each action or interpretation that can be used by the HITSI Application when deciding what action should be taken from sensor inputs. There is no limit to the amount of actions a user can implement

Example:

```
<Action>
        <ActionName>NextSlide</ActionName>
        …
</Action>
```

<ActionID>

Description: This field is used to uniquely identify each Action by an ID number

Example: <ActionID>1</ActionID>

<ActionName>

Description: This field contains the name of the Action that may be taken if the HITSI Application is able to match sensor input to the interpretation information contained in the child fields of Action.

Example: <ActionName>NextSlide</ActionName>

<WParam> and <LParam>

Description: wParam and lParam are used to identify what command is to be issued to the third party application. wParam is split into high-order word and low-order word. The high-order word specifies the notification code if the message is from a control. A control can be either an accelerator (value of wor dis 1) or a menu (value of word is 0). The low-order word specifies the identifier of the menuitem control or accelerator. The lParam is the handle to the control sending the message. If there is no control then this value is NULL.

Example:

<WParam> 00010189</WParam>

<LParam>0x00000000</LParam>

<FireCond>

Description: The children fields of this field contains information relating to one action or interpretation that can be used by the HITSI Application when deciding what action should be taken from sensor inputs. A maximum of one FireCond node can be included for each Sensor (Max of Eight)

Example:

```
<FireCond>
        <SensorID>0</SensorID>
        …
</FireCond>
```

<Equation>

Description: This field is used to identify whether a successful sensor reading is “Greater Then” or “Less Then” the sensor value identified in the “Value” field.

Example: <Equation>Greater Then</Equation>

<Value>

Description: This field contains the value that when combined with the “Equation” field is used to determine if a sensor reading is a successful sensor reading. (note this value must be an integer)

Example: <Value>250</Value>

4.3 Design Decisions

This section approaches the issues that arose in chapter 3 and also some issues that arose in the implementation of the HITSI application. The sub sections below go into greater detail to explain what methods and rules were used in the implementation of Time Windows, Interpretations and System Messages in the HITSI application.

4.3.1 Time Window Operation

During operation it was discovered that sensor inputs meant for a single interpretation could still be spread across two separate time windows. This circumstance can occur either when a user starts their sensor input at the end of a time window, which collapses before the sensor input is complete (1) or when a user takes too long to input all sensor readings (2). The separate interpretations that result from this are clearly undesired by the user.

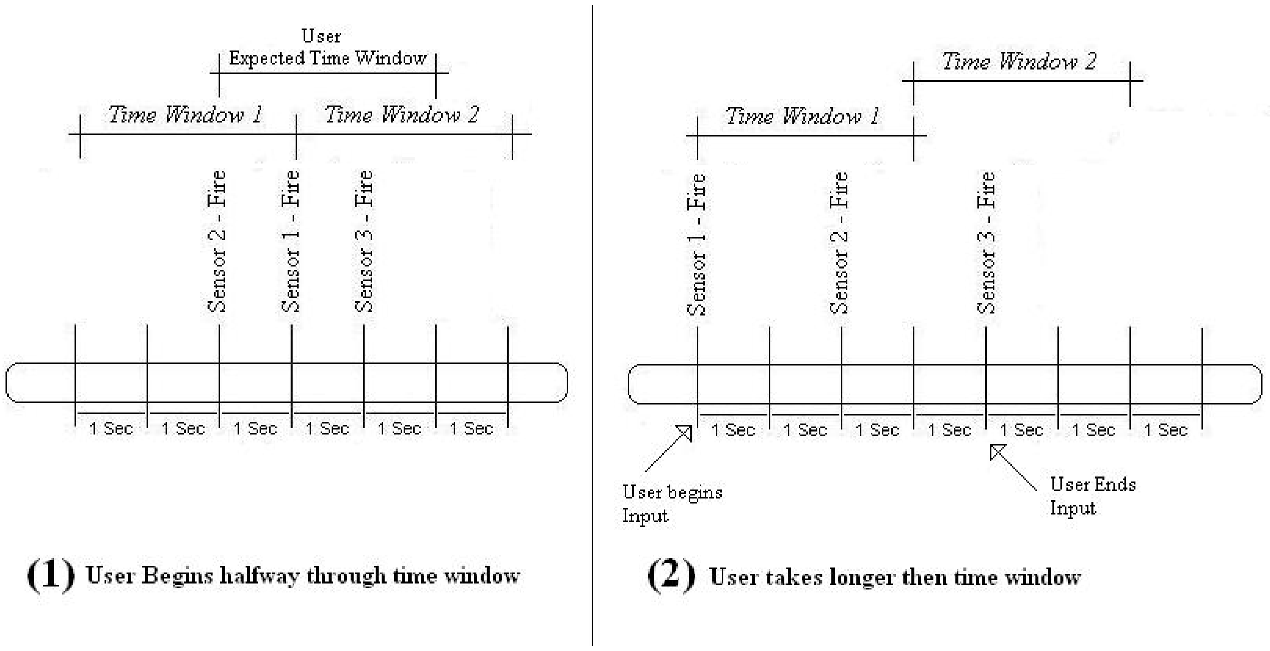


Figure 28: User Input Issues With Time Window

To tackle the first issue of a user beginning sensor inputs halfway through a time window the static time window was modified (refer to chapter 3.2.4). The time window was modified to only begin a time window when a sensor reading occurs outside of any existing time window. The diagram below illustrates that the time window does not begin until a sensor reading has occurred. The time window exists until the time period expires and any future sensor readings after this event will begin the process again.

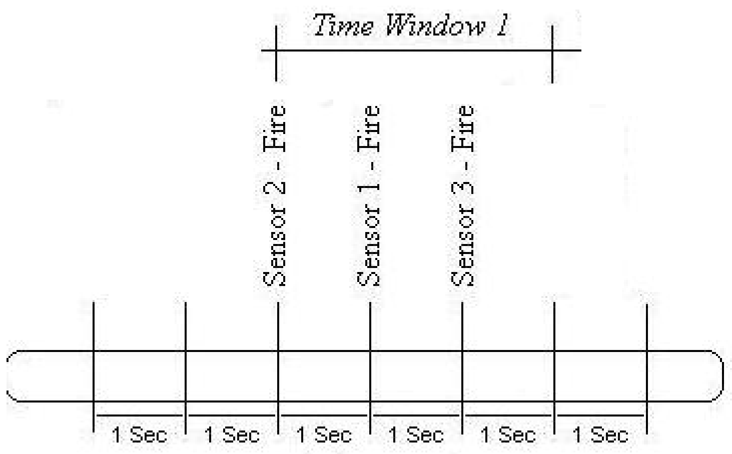


Figure 29: Solution to Figure 9 (1)

The issue that arises from a user not being able to complete all sensor inputs in the designated time window is overcome by allowing for a user adjustable time window. The user adjustable time window is implemented in the XMLData.xml file. This value (in the <waittime> element) is read into the HITSI Application during runtime at the time when the user specifies which application (Existing in XMLData.xml) they will use. If the user finds that they are unable to input the required sensor readings in the allocated time period (Time Window) for a successful interpretation they are able to modify the time window value to allow enough time for the required sensor inputs.

The following pseudo code describes the process outlined in Figure 10 and includes the adjustable time window as implemented in the HITSI Application prototype. Both process run in parallel and are able to operate independently.

Pseudo code for Sensor Inputs

```
1)      Wait for sensor input
2)      Sensor Input Received
3)      If timer (TimeWindow) is active goto 5
4)      Begin timer  (Time Window)
5)      Store sensor input details for further use
6)      Goto 1
```

Pseudo Code for Timer

```
1)      Wait for timer activation
2)      Activate Timer
3)      When timer reaches time window value (Time Window Collapse)
            run interpretations
            stop and reset timer
```

4.3.2 Interpretation Refinement

The concept of Interpretations was introduced in chapter 3. This section looks closer at the implementation of interpretations in the HITSI application. The section covers where the information for interpretations is obtained and pseudo code for sensor interpretations

4.3.2.1 Runtime Interpretations

While interpretation information is stored in the XMLData.xml file there were two options in how to use this information. The first option was to lookup the required information out of the XML file when the information was required during runtime.

This approach requires little design structure in the application as data would never be stored and thus no internal classes are required to model the information. This way of accessing the information would also provide the possibility of changing interpretation information during runtime. Although changing interpretation information during runtime might seem like a feature it is difficult to implement, as it would be very hard to change these settings using the sensor inputs. Alternatively if the keyboard was used it would contradict the aims of the HITSI Application.

The second approach required the design of an interpretation class inside of the HITSI Application that is used to store all related interpretation information that is read in only at the start of execution. To lessen the amount of data stored during runtime, application interpretation information is stored only after the users' third party application is selected. This refined data storage means less time when resolving interpretations and less memory used to store interpretation information during runtime. Because of the performance benefits associated with this second option it was chosen as the method that would be used.

4.3.2.2 Interpretation Decisions

The HITSI Application uses a mix of priorities and scoring system. When a time window collapses and an interpretation is to take place the Phidgets app goes through the following sequence of events

```
1)      Time Window Collapses
2)      Start Interpretation
3)      While more interpretations exist select next interpretation
4)      While more Fire Conditions exist in Interpretations select next Fire Condition
5)      Add one to current interpretation Fire Conditions Count
6)      If Fire Condition equation equals “Greater Then”:  then
            If Fire Condition Value is greater then Sensor Value for matching sensor Ids then
                add one to interpretation Fire Conditions Correct Count
        Else If Fire Condition equation equals “Less Then” then
            If Fire Condition Value is less then Sensor Value for matching sensor Ids then
                add one to interpretation Fire Conditions Correct Count
        Else:
            Assume a mistyped XML entry and ignore Fire Condition
7)      If more Fire Conditions in Interpretation go to 4
8)      If Fire Conditions Count equals Fire Conditions Correct Count
            Interpretation Match  = Fire Conditions Count * 100
        Else
            Interpretation Match = (Fire Conditions Correct Count / Fire Conditions Count) -
                                   .74 * (Fire Conditions Count * 10)
9)      If interpretation “match” value is greater than the stored “selected Match” value then
        store the current interpretation as the “Selected Match” Interpretation
10)     If more Interpretations exist go to 3
```

At step three and four it is assumed that all relevant information for the specific user and application have been processed into the applications data structures.

The scoring system is designed to allow the following:

- Any interpretation with a perfect match (all Fire Conditions are met) outscores any interpretation with a partial match
- Any interpretation with a perfect match with more fire conditions will outscore an interpretation with a perfect match and less Fire Conditions
- A partial match will only be the best match if no perfect match exists and the partial match has at least 75% of the Fire Conditions met
- In the absence of a perfect match and multiple partial matches the selected partial match will be decided taking the percentage of Fire Conditions met and also the total number of fire conditions for each interpretation into account.
- When two interpretations score the same match value the interpretation that is higher in the XML document (and therefore first in the interpretation array) will be chosen
- An interpretation that does not have 75% or more Fire Conditions met is to be ignored

The scoring system above attempts to provide an intuitive way of interpreting sensor inputs. An interpretation with more fire conditions met will be chosen before an interpretation with less fire conditions met. The 75% match value was chosen to allow the user a small margin of error when attempting sensor inputs. 75% means that an interpretation with 3 or fewer fire conditions requires 100% of these fire conditions to be met but with four fire conditions only three need to be met for the interpretation to qualify for a selection.

4.3.3 System Messages

The implementation of the HITSI application makes use of Microsoft Windows WM_COMMAND messages and the WIN32.SendMessage function in C#. As shown in Figure 19 in Chapter 3 the HITSI application is able to control the third party application through the use of the wm_command message issued by the HITSI application to the OS.

To successfully issue the wm_command to the correct third party application the application needs to be identified. To identify an application window two variables need to be identified. The first variable is the class name of the window in which the third party application runs and the second is the title of the window in which the third party application runs. The HITSI application obtains this information from the DB (refer to section 4.2.4) and uses the Win32.FindWindow function to obtain a handle to the designated window. With this handle the HITSI application is able to use the WIN32.SendMessage function to send the wm_command message.

The wm_command message contains two variables. They are the lParam and wParam. wParam and lParam are used to identify what command is to be issued to the third party application. wParam is split into high-order word and low-order word. The high-order word specifies the notification code if the message is from a control. A control can be either an accelerator (value of word is 1) or a menu (value of word is 0). The low-order word specifies the identifier of the menuitem control or accelerator. The lParam is the handle to the control sending the message. If there is no control then this value is NULL.

The wm_command message emulates the command that is issued when a menu item is selected, a key is pressed or an accelerator function fired. An accelerator function is fired when a combination of key presses is mapped to an action (for example holding the ctrl and s keys at the same time usually maps to an accelerator save function)

4.4 Summary

Chapter 4 focused on refining the design of the HITSI application. The concepts behind Time Windows and Interpretations were defined as used in the HITSI application implementation. The technologies used in the implementation of the HITSI application were discussed as well as the use of system messages. The HITSI application design was formalised with the use of Sequence, Class and Component UML diagrams. The design of the database and how the data from the database is used is also explored.

Chapter 5 HITSI Application Evaluation

This chapter looks at measuring the success of the HITSI application implementation. Measurements include performance, accuracy and soundness. The HITSI application is also introduced as a prototype and the limitations of the prototype are discussed.

5.1 Performance

The performance of the HITSI Application can be analysed by discerning how quickly a set of human sensor inputs can be translated into a correct interpretation. The time period between sensor input and interpretation is a direct relation to the time window in the HITSI Application.

The HITSI Application has a recommended time window of half a second or greater. Half a second is the smallest time window that allows for average usage of sensor inputs during normal operation. While half a second is a small amount of time it is not responsive enough to be effectively used with high interaction applications such as some computer games. Half a second however is quite a short amount of time to make several sensor inputs into the HITSI Application.

A test was performed to deduce how many readings the Phidgets Interface Kit could take in a five second interval. The results for a motion sensor were 37, 43, 43, 42, 39 inputs in a five second period. This averages out to approximately 8 inputs per second. The amount of inputs that are received, processed and forwarded by the Phidgets Interface Kit varies depending on what sensors are attached. A second identical test was performed using a force sensor instead of the motion sensor. This test revealed results of 115, 106 119, 94 and 108 inputs received during a five second period. This averages out to approx 21 inputs per second (much higher then the motion sensor). As can be seen by the results obtaining enough sensor inputs during a time window is not restricted by the speed in which they can be recorded but rather how quickly a user can enter their multiple sensor readings accurately.

Another performance issue that may have restricted the time window to a minimum of half a second is the time in which it takes collectively for the application to interpret all the sensor readings and for the third party application to perform the requested action. I performed a test that took the time before the interpretation function was called and took another reading after the interpretation had been decided and the command had been sent to the third party application. The time it took for this process to complete was .09375secs. This time is virtually unrecognizable in the context of human perception and input. The amount of interpretations in this test was 249, which is many more then would be used in normal operation.

5.2 Accuracy

One of the reasons the Phidgets Interface Toolkits (PIKs) were used in the development of the HITSI Application prototype was the confidence and commercial success that these sensor packages have achieved. However whilst the PIKs are proven to be accurate in there sensor readings, human interactions through these sensors is never a simple thing.

The PIK is able to read motion, light changes, force changes and many other things but not interpret these results. How does the system know the difference between the user activating the motion sensor or a person walking past the active area? While these interpretations of actions are instantly clear to the average person how do you make the system understand this observation?

The use of a multiple sensor inputs being grouped together into a time window allows for some undesired sensor inputs to be disregarded but a 100% accuracy in sensor input interpretations is still something that needs to be developed.

In the HITSI Application prototype the introduction of a time window and the concept of interpretation were used to address accuracy issues. The Time Window attempted to allow a reasonable period of time in which to gather enough sensor input to correctly evaluate what the user intended. Similarly the concept of interpretation was introduced to allow these sensor readings to be analysed and subjugated to a set of criteria that attempts to understand what the user wished to achieve.

5.3 Soundness

In order for a user to operate the HITSI Application confidently and utilise all its features the user must be aware of how interpretation decisions are made. Without the decision making knowledge a user may not be able to a) Modify the XMLData.xml file to fulfil their specifications b) Understand why particular interpretations are constantly chosen or disregarded c) Correctly enter interpretation information that will operate in the way they desire.

Currently no standard is in place that defines a particular decision tree or set of rules that is used in sensor-based interpretations. The aims of these rules are to be intuitive and eventually so well designed and used that users need not know they exist or are so comfortable in the knowledge that they are second nature.

The HITSI application design attempted to implement ideas that were easy to understand and be as intuitive as possible for the user. The XML database is structured in such a way as to provide users with information in a format that is easy to read, modify and understand. Similarly the rules involved in the interpretation process are designed to be practical and intuitive.

5.4 One "Serious Large" Application

One of the applications the HITSI Application prototype was developed for was the possibility of providing an interactive slideshow presentation during a University open day. The HITSI Application would be set up in a regular room with the Phidgets Interface Kit including 2 Motion sensors and 2 Force Sensors. The layout of the room would appear as in the diagram below.

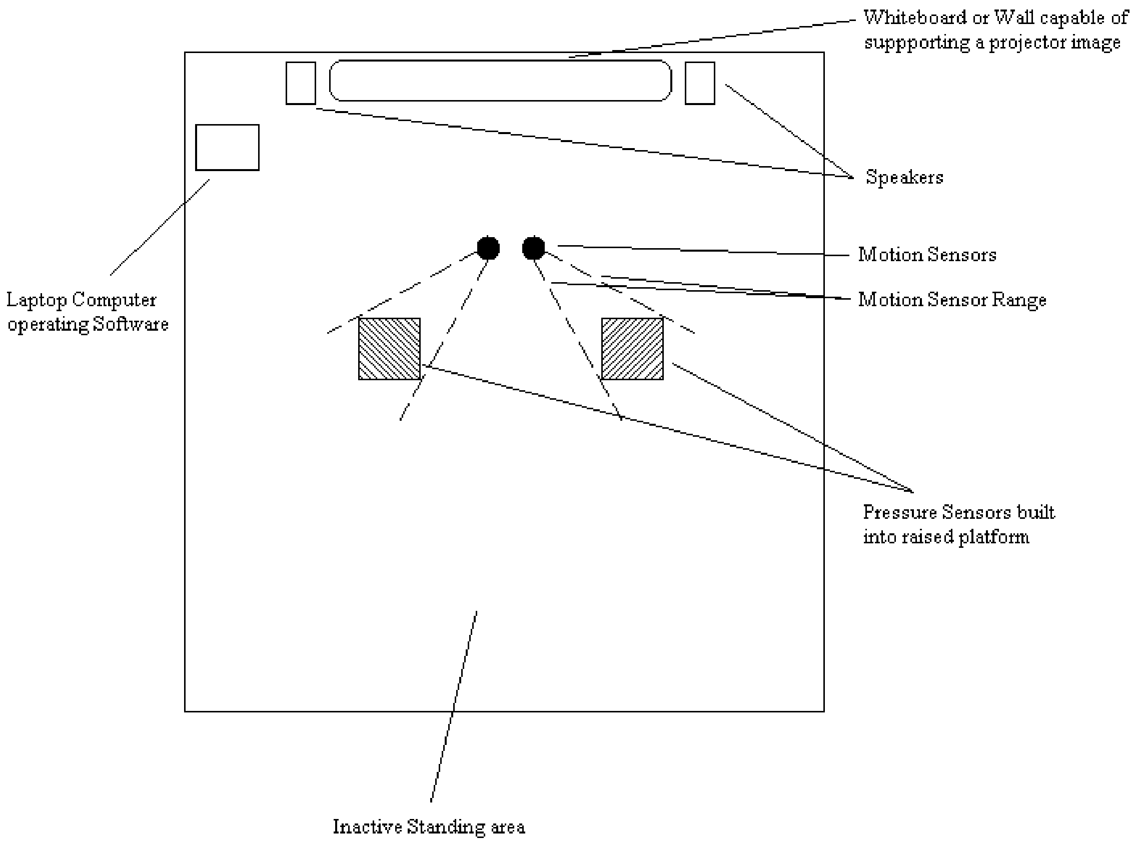

Whiteboard or Wall capable of suppporting a projector image
Speakers
Motion Sensors
Motion Sensor Range
Laptop Computer operating Software
Pressure Sensors built into raised platform
Inactive Standing area

Figure 30: HITSI Demonstration Setup

Prospective students enter the room. Whilst nobody is standing on the raised platforms any activity in the room is ignored. As soon as one or both platforms are occupied a media file is played that contains information about the computer science facilities at Latrobe Bundoora. Whilst the platforms are occupied any exaggerated or speedy movements are picked up by the motion sensors and the slide show progresses to the next slide. Once both platforms are empty again the media file is stopped ready to be activated the next time the platform is occupied and the slideshow progress halts until the same circumstances. This Presentation requires no person watching over it and can run for the entire duration of the day.

The source code for the HITSI application prototype implantation can be found in Appendix A and the XMLData file used can be found in Appendix B

5.5 Limitations

This section looks at the limitations that were observed when the HITSI application prototype was implemented. The limitations include those that are imposed by the HITSI application design and those imposed by the new form of computer interaction.

5.5.1 Acquiring System Messages and Application Identification

A limitation for users is the inability to acquire new system messages and application identification quickly and easily. Currently the way to acquire these messages is through the use of Spy++. Even if a user has access to the Spy++ application the information required is hard to understand and extract. Whilst obtaining application identification is relatively easy for someone who has had experience with Spy++ previously the process in which to obtain useful and relevant application information and system messages would be difficult and tedious for a new user.

5.5.2 Eight Sensor Inputs

The HITSI Application prototype only allows up to eight sensors attached to the Phidgets Sensor Kit. While eight sensors should be more then enough for its designed usage future implementations may require the use of more. The Phidgets Interface Kits allow for the attachment of more Interface kits through the use of the two-port USB hub. Each additional Phidgets kit allows for an extra eight sensor attachments. Additionally the HITSI Application would require updating to allow for the extra sensor inputs. The XML file/format that stores information relating to sensor identification would still be valid with the addition of the extra Phidgets Interface kit as the new sensor would simply adopt a new sensor ID.

5.5.3 User Inexperience

As sensor input is an emerging form of interaction with computer systems users are not experienced and may not be comfortable yet using the technology involved in sensor input. Users may not even be aware of the technology. This means that every new user will undergo a learning curve in order to understand and operate the HITSI application.

Inexperienced users will be unable to create or modify the XML file in order to customize interpretation information. Users without a good understanding of how the rules, equations and ideas work in respect to interpretations will be unable to achieve the correct mapping of sensor inputs to application actions. To learn this information users will need to have access to information explaining the interpretation rules and also how to create individual preferences and XML files.

5.5.4 Standards

As multiple sensor input is an emerging form of interaction with computer applications and most technologies involved may be new to users there are still no standards to define human based sensor input in this form of application.

Concepts such as Fire Conditions, Time Windows and Interpretations all contain ideas and approaches to achieving issues associated with human based multiple sensor input. Because the ideas and approaches in these concepts are new and not the only possible concepts available, users could be at a disadvantage to understand each new attempt at creating a sensor based human interaction computer application.

5.6 Summary

This chapter looked at the performance of the HITSI application implementation in the form of limitations, accuracy and soundness.

This chapter looks at measuring the success of the HITSI application implementation. Measurements include performance, Accuracy and Soundness. The HITSI application is also introduced as a prototype and the limitations of the prototype are discussed.

Chapter 6 Conclusion

This chapter looks at the whole thesis up to this point and outlines not only what has been learnt and achieved but also the issues that arose including any solutions that were created and the limitations the new application and design have imposed. From this information an evaluation is performed and future possibilities and advancements are considered.

6.1 Summary

This thesis has described my rational and process in creating an application that is capable of sensor based human interaction from defining the aims and my approach through to evaluation of the solution I developed.

The literature review looked at what sensor based human interaction systems, standard and architectures had been attempted. From this information I was able to further clarify the aims of my prototype application.

I was able to create a model of the HITSI application through the use of UML diagrams and also discuss the issues that arose from attempting to design the HITSI application in this way. Time Windows and mapping sensor inputs to contexts through the use of interpretation were the main issues discussed.

I was also able to design specific classes and components through the use of UML and explain how each diagram related to achieving an aim of the HITSI application. From these designs I created a Prototype of the HITSI application. The Evaluation of the prototype is discussed highlighting limitations, performance, accuracy and soundness.

The use of this information should provide a person wishing to attempt the creation of an application that supports sensor based human interaction with a computer application the base knowledge to not only design the application but to improve it through the understanding of the issues generated through this experience.

6.2 Future Work

The aim of the HITSI Application was to create a sensor based human interaction application. The HITSI application achieved this through the use of several concepts including Time Windows and Interpretations. Although the HITSI prototype was a success in meeting this aim several limitations arose as described in section 5.5. These limitations bring forward what direction future work could take.

In addressing User inexperience and lack of operational knowledge work could be directed in attempting to make the operation of the sensor input and resultant interpretation more intuitive to the user. By increasing the intuitiveness of the application the learning curve required and difficulty in operating an application such as the HITSI application is dramatically reduced.

As more and more ideas are developed in attempts to create an efficient and accurate human sensor based sensor interaction computer application benefits and disadvantages for each will arise. A study into the exact benefits and disadvantages of each approach could result in a standout method or idea that could be introduced as a standard in the future.

Work could also be directed at the way a third party application is controlled. There is potential to create a standard language or platform in which all applications could interact with each other. A simple and easy way to identify windows could be introduced or even a standard document released with each application that identifies application identification and lists the way in wich a third party application can communicate.

Future work could also include introducing a way in which more then one, third party application can be controlled during the one execution of the HITSI application. Studies could also be aimed at formalising a set of standard rules that interpretations will have to meet.

References

1. Weiser, M. *Nomadic Issues in Ubiquitous Computing*. in *Nomadic 96' Conference*. 1996.
2. Weiser, M., *Some Computer Science Issues in Ubiquitous COmputing.* Communications of the ACM, 1993. Vol. 37(Issue. 7): p. 75-84.
3. Abowd, G.D., *Classroom 2000: An experiment with the instrumentation of a living Education Environment.* IBM Systems Journal, 1999. Vol. 38(Issue. 4): p. 508-530.
4. Hong, J.I., *Minimizing Security Risks in Ubicomp Systems.* Computer, 2005. Vol. 38(Issue. 12): p. 118-119.
5. Krikke, J., *T-Engine: Japan's ubiquitous computing architecture is ready for prime time.* Pervasive Computing, 2005. Vol. 4(Issue. 2): p. 4-9.
6. Greenberg, S., *Phidgets: Easy Development of Physical Interfaces through Physical Widgets.* Proceedings of the UIST 2001 14th Annual ACM Symposium on User Interface Software and Technology, 2001: p. 209-218.
7. Raman, T.V. *User Interface Principles for Multimodal Interaction*. in *MMI Workshop*. 2003. CHI 2003: IBM Research.
8. Landay, J., *Invisible Computing Activities.* Microsoft Research Summer InstituteTechnologies of Invisible Computing 1999, 1999.
9. *Navy.Mill.* http://www.news.navy.mil/view_single.asp?id=3523.
10. Oviatt, S., *Handbook of Human-Computer Interaction (Multimodal Interfaces Chapter)*, ed. M. Interfaces. 2002.
11. Corradini, A., et al., *Multimodal Input Fusion in Human-Computer Interaction - On the Example of the NICE Project.* NATO Science Series, III: Computer and Systems Sciences, 2005. Vol 198: p. 223-234.
12. Dey, A.K. and G.D. Abowd, *A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications.* Human Computer Interaction Journal, 2001. Vol. 16(Issue. 2): p. 97-166.
13. Flores-Mendez, R.A., *Towards a Standardization of Multi-Agent System Frameworks.* Crossroads, 1999. Vol. 5(Issue. 4): p. 18-24.
14. *Intel.* http://www.intel.com/technology/computing/applications/avcsr.htm.
15. Franklin, D., *The Intelligent Classroom.* IEEE, 1999. Vol. 14(Issue. 5): p. 2-5.
16. Benerecetti, M., P. Bouquet, and B. M, *Distributed Context-Aware Systems.* Human Computer Interaction, 2001. Vol. 16: p. 213-228.
17. Schmidt, A., M. Beigl, and H.W. Gellersen, *There is more to context than location.* Computer and Graphics Journal, 1999. Vol. 23(Issue. 6): p. 893-902.
18. Lieberman, H. and T. Selker, *Out of context: Computer systems that adapt to, and learn from, context.* IBM SYSTEMS JOURNAL, 2000. Vol. 39(Numbers 3 & 4).
19. Singh, A., *Survey of Context Aware Frameworks - Analysis and Criticisms.* UNC, 2006. Version 1.
20. *Future Computing Environments Group.* http://www-static.cc.gatech.edu/fce/projects.html.
21. Kaenampornpan, M. *Bringing Context Awareness Together*. in *Second International Conference on Pervasive Computing*. 2004.
22. Chen, H., T. Finin, and A. Joshi, *An Intelligent Broker for Context-Aware Systems.* Adjunct Proceedings of Ubicomp 2003, 2003: p. 183-184.
23. Gu, T., H.K. Pung, and D.Q. Zhang, *A middleware for building context-aware mobile services.* IEEE 59th, 2004. Vol. 5: p. 2656-2660.
24. Ayenew, B., *Merging Ontologies in the Context Mediation Framework (Thesis)*. 2002.
25. Kawsar, F., K. Fujinami, and Tatsuo Nakajima. *Prottoy: A Context Aware Application Framework*. in *UCS 2004*. 2004. Keio University Mita Campus, Tokyo, Japan.
26. *Microsoft Corporation*, http://research.microsoft.com/easyliving/.

Appendix A

HITSI Application source code

Main Screen

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Xml;
using System.Xml.XPath;

namespace Phidgets
{


    public partial class Main : Form
    {

        Errors errors = new Errors();

        public Main()
        {
            InitializeComponent();
        }

        private void Main_Load(object sender, EventArgs e)
        {
            XmlDocument docXMLFile = new XmlDocument();
            try
            {
                docXMLFile.Load("XMLdata.xml");
            }
            catch {
                errors.Error(3);
                return;
            }

            XmlNodeList users;
            XmlElement root = docXMLFile.DocumentElement;
            users = root.SelectNodes("/Config/User/UserName");

            foreach (XmlNode user in users)
            {
               UserCombo.Items.Add(user.InnerText);
            }

        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                Config config = new Config(UserCombo.SelectedItem.ToString());
                config.Show();
            }
            catch { errors.Error(1); }
        }

        private void label1_Click(object sender, EventArgs e)
        {

        }

        private void button2_Click(object sender, EventArgs e)
        {
            string tempFileName;
            string tempApp = textFile.Text;
            XmlDocument docXMLFile = new XmlDocument();

            try
            {

                docXMLFile.Load("XMLdata.xml");
            }
            catch {
                errors.Error(3);
                return;
            }


            XmlNode extensionnode;
            XmlElement root = docXMLFile.DocumentElement;
            try
            {
                extensionnode =
root.SelectSingleNode("/Config/User[./UserName='" +
UserCombo.SelectedItem.ToString() + "']/App[./AppName='" +
appCombo.SelectedItem.ToString() + "']/FileExtension");
            }
            catch {
                errors.Error(2);
                return;
            }

            try
            {
                int first = textFile.Text.LastIndexOf("\\") + 1;
                int last = textFile.Text.LastIndexOf(extensionnode.InnerText) -
1;

                tempFileName = textFile.Text;
                tempFileName = tempFileName.Substring(first, last - first);
            }
            catch {
                errors.Error(4);
                return;
            }

            phidgets phidget = new phidgets(UserCombo.SelectedItem.ToString(),
appCombo.SelectedItem.ToString(), tempFileName);

            phidget.Show();

        }

        private void UserCombo_SelectedIndexChanged(object sender, EventArgs e)
        {
            XmlDocument docXMLFile = new XmlDocument();
            try
            {

                docXMLFile.Load("XMLdata.xml");
            }
            catch
            {
                errors.Error(3);
                return;
            }

            XmlNodeList apps;
            XmlElement root = docXMLFile.DocumentElement;
            apps = root.SelectNodes("/Config/User[./UserName='" +
UserCombo.SelectedItem.ToString() + "']/App/AppName");


            appCombo.Items.Clear();
            foreach (XmlNode app in apps)
            {
                appCombo.Items.Add(app.InnerText);
            }


        }

        private void openFile_Click(object sender, EventArgs e)
        {
            XmlDocument docXMLFile = new XmlDocument();
            try
            {

                docXMLFile.Load("XMLdata.xml");
            }
            catch{
                errors.Error(3);
                return;
            }


            XmlNode extension;
            XmlElement root = docXMLFile.DocumentElement;

            openFileDialog1.RestoreDirectory = true;

            extension = root.SelectSingleNode("/Config/User[./UserName='" +
UserCombo.SelectedItem.ToString() + "']/App[./AppName='" +
appCombo.SelectedItem.ToString() + "']/FileExtension");


            openFileDialog1.InitialDirectory = @"C:\";
            openFileDialog1.Title = "Select a File";
            try
            {
                openFileDialog1.Filter = "" + appCombo.SelectedItem.ToString()
+ " Files|*." + extension.InnerText;
                openFileDialog1.FilterIndex = 1;
            }
            catch { }

            if (openFileDialog1.ShowDialog() != DialogResult.Cancel)
                textFile.Text = openFileDialog1.FileName;
        }

    }
    public class Errors
    {

        private string Message, Title;

        public void Error(int ErrorNum)
        {

            XmlDocument docXMLFile = new XmlDocument();
            try
            {

                docXMLFile.Load("Errors.xml");
            }
            catch
            {
                MessageBox.Show("Could Not Find Erros.xml", "File Not Found");
            }


            XmlNode Action;

            XmlElement root = docXMLFile.DocumentElement;
            Action = root.SelectSingleNode("/Errors/Error[./ErrorID='" +
ErrorNum + "']");

            Action.RemoveChild(Action.FirstChild);
            Message = Action.FirstChild.InnerText;
            Action.RemoveChild(Action.FirstChild);
            Title = Action.FirstChild.InnerText;
            MessageBox.Show(Message, Title);
        }
    }
}
```

Sensor Monitoring Screen

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Phidgets;
using Phidgets.Events;
using System.IO;
using System.Xml;
using System.Runtime.InteropServices;


namespace Phidgets
{

    public partial class phidgets : Form
    {

        Errors errors = new Errors();
        //private System.Int32 iHandle;

        private XmlNode actionm, tempNode;

        private XmlDocument docXMLFilem = new XmlDocument();

        ThirdParty ThirdPartyApp;

        string tempClass, tempFirstHalf, tempLastHalf, tempFileName;

        int TimeWindow;
        int[] Sens = new int[8];

        //ifkit: is the instance of the phidgets software
        InterfaceKit ifkit;

        //clock: is the timer used to judge window size
        Timer Clock;


        //ActionInterpretor:  is the instance of the class that does all of the
        //aciton interpretations
        Action_Interpretor ActionInterpretor = new Action_Interpretor();

        // This is the constructor to the sensor monitoring screen
        //This initalizes the timer, the phidgets kit and the thirdParty
        public phidgets(string tempUser, string tempApp, string tempFilename)
        {

            try
            {
                docXMLFilem.Load("XMLdata.xml");
            }
            catch
            {
                errors.Error(3);
                return;
            }

            XmlElement rootm = docXMLFilem.DocumentElement;

            Clock = new Timer();

            InitializeComponent();


            Applbl.Text = tempApp;
            Userlbl.Text = tempUser;
            FileNamelbl.Text = tempFilename;


            actionm = rootm.SelectSingleNode("/Config/User[./UserName='" +
Userlbl.Text + "']/App[./AppName='" + Applbl.Text + "']");

            tempNode = actionm.SelectSingleNode("ClassName");
            tempClass = tempNode.InnerText;

            tempNode = actionm.SelectSingleNode("FirstHalfWindowName");
            tempFirstHalf = tempNode.InnerText;

            tempNode = actionm.SelectSingleNode("LastHalfWindowName");
            tempLastHalf = tempNode.InnerText;
            tempNode = actionm.SelectSingleNode("FileName");

            if (tempNode.InnerText == "true")
            {
                tempFileName = tempFilename;
            }
            else
            {
                tempFileName = "";
            }

             ThirdPartyApp = new ThirdParty(tempClass, "" + tempFirstHalf +
tempFileName + tempLastHalf);

            tempNode = actionm.SelectSingleNode("WaitTime");
            try
            {
                TimeWindow = Convert.ToInt16(tempNode.InnerText);
            }
            catch {
                errors.Error(6);
                TimeWindow = 1000;
            }


            Clock.Interval = TimeWindow;
            Clock.Start();
            Clock.Tick += new EventHandler(Timer_Tick);

        }
        //This is the defining of the phidgets kit and the methods used to
monitor he sensor inputs
        private void phidgets_Load(object sender, EventArgs e)
        {
            ifkit = new InterfaceKit();

            ifkit.SensorChange += new
SensorChangeEventHandler(ifkit_SensorChange);
            ifkit.open();
            ActionInterpretor.fillInterpretations(Userlbl.Text, Applbl.Text);
        }

        //This is the method that moniters the sensor inputs of the phidgets
interface kit
        void ifkit_SensorChange(object sender, SensorChangeEventArgs e)
        {
            if (Clock.Enabled == false)
            {
                Clock.Start();
            }
            switch (e.Index)
            {
                case 0:
                         ActionInterpretor.addAction(0, e.Value);
                         Sens[0] = e.Value;

                    break;
                case 1:
                        ActionInterpretor.addAction(1, e.Value);
                        Sens[1] = e.Value;
                    break;
                case 2:
                        ActionInterpretor.addAction(2, e.Value);
                        Sens[2] = e.Value;
                    break;
                case 3:
                        ActionInterpretor.addAction(3, e.Value);
                        Sens[3] = e.Value;
                    break;
                case 4:
                        ActionInterpretor.addAction(4, e.Value);
                        Sens[4] = e.Value;
                    break;
                case 5:
                        ActionInterpretor.addAction(5, e.Value);
                        Sens[5] = e.Value;
                    break;
                case 6:
                        ActionInterpretor.addAction(6, e.Value);
                        Sens[6] = e.Value;
                    break;
                case 7:
                        ActionInterpretor.addAction(7, e.Value);
                        Sens[7] = e.Value;
                    break;

            }

        }

        //This method fires when the timer reaches the time window length.
        //This method runs the interpretations and calls the third party class
to action
        public void Timer_Tick(object sender, EventArgs eArgs)
        {
            int Command = 0;
            DateTime first;
            DateTime second;

            first = DateTime.Now;
            ActionInterpretor.runInterpretor();

            Clock.Stop();


            try
            {

                CurrentActionlbl.Text = ActionInterpretor.getAction();
            }
            catch { }
            try
            {
                if (ActionInterpretor.getAction() != "" &&
ActionInterpretor.getAction() != null)
                {
                    if (ActionInterpretor.getCommand() == "keypress")
                    {
                        Command = Win32.WM_KEYPRESS;

                    }
                    else
                    {
                        Command = Win32.WM_COMMAND;
                    }
                    ThirdPartyApp.sendMessage(ActionInterpretor.getlParam(),
ActionInterpretor.getwParam(), Command);
                }
                else
                {
                }
            }
            catch { }
            second = DateTime.Now;

            ActionInterpretor.clearActions();
            for (int i = 0; i < 8; i++)
            {
                ActionInterpretor.addAction(i, Sens[i]);
            }
            if (Alert.BackColor == Color.Blue)
                Alert.BackColor = Color.Black;
            else
                Alert.BackColor = Color.Blue;

        }

        //This class models the actions montiored by the sensors
        //Sensor inputs are stored in this class
        public class ActionClass
        {
            private double SensorValue;

            public ActionClass()
            {
                return;
            }

            public double getSensorValue()
            {
                return SensorValue;
            }

            public double getTimeStamp()
            {
                return 0;
            }

            public void setSensorValue(double tempSensorValue)
            {
                SensorValue = tempSensorValue;
                return;
            }


        }

        //This class is used to store Fire OCndition information.  The
information
        //is read into this class at the beginning of execution and extracted
when
        //interpretations take place
        public class Fire_Cond
        {
            private int SensorID = 0;
            private string Equation;
            private double Value;
            private int Match = 0;

            public Fire_Cond()
            {

            }

            public int getSensorID()
            {
                return SensorID;
            }

            public string getEquation()
            {
                return Equation;
            }

            public double getValue()
            {
                return Value;
            }

            public int getMatch()
            {
                return Match;
            }

            public void setEquation(string newEquation)
            {
                Equation = newEquation;
            }

            public void setValue(double newValue)
            {
                Value = newValue;
            }
        }

        //This class models information information.  This class contains
        // the data on fire conditions, third party application commands and
match values.
        public class Interpretation
        {
            private string INT_Action, INT_Command;
            private int INT_wParam, INT_lParam;
            private int INT_ActionID;
            private double Match, Matches;


            private Fire_Cond[] FireConds;
            int MAX_INTERPRET_COUNT = 8;

            public Interpretation(string tempUser, string tempApp, string
Action, int tempwParam, int templParam, int tempActionID, string tempCommand)
            {
                INT_Action = Action;
                INT_lParam = templParam;
                INT_wParam = tempwParam;
                INT_ActionID = tempActionID;
                INT_Command = tempCommand;

                FireConds = new Fire_Cond[MAX_INTERPRET_COUNT];
                for (int i = 0; i < MAX_INTERPRET_COUNT; i++)
                {
                    FireConds[i] = new Fire_Cond();
                }
                XmlDocument docXMLFile = new XmlDocument();
                try
                {

                    docXMLFile.Load("XMLdata.xml");
                }
                catch
                {
                    MessageBox.Show("Could not find XMLdata.xml", "Missing
File");
                    return;
                }

                XmlNodeList firecondlist;
                XmlNode tempNode;
                XmlElement root = docXMLFile.DocumentElement;

                firecondlist = root.SelectNodes("/Config/User[./UserName='" +
tempUser + "']/App[./AppName='" + tempApp + "']/Action[./ActionName='" + Action
+ "']/FireCond");

                foreach (XmlNode firecond in firecondlist)
                {
                    switch (Convert.ToInt16(firecond.FirstChild.InnerText))
                    {
                        case 0:
                            tempNode = firecond.SelectSingleNode("Equation");
                            FireConds[0].setEquation(tempNode.InnerText);

                            tempNode = firecond.SelectSingleNode("Value");

FireConds[0].setValue(Convert.ToInt16(tempNode.InnerText));

                            break;
                        case 1:

                            tempNode = firecond.SelectSingleNode("Equation");
                            FireConds[1].setEquation(tempNode.InnerText);

                            tempNode = firecond.SelectSingleNode("Value");

FireConds[1].setValue(Convert.ToInt16(tempNode.InnerText));
                            break;

                        case 2:
                            tempNode = firecond.SelectSingleNode("Equation");
                            FireConds[2].setEquation(tempNode.InnerText);

                            tempNode = firecond.SelectSingleNode("Value");

FireConds[2].setValue(Convert.ToInt16(tempNode.InnerText));
                            break;

                        case 3:
                            tempNode = firecond.SelectSingleNode("Equation");
                            FireConds[3].setEquation(tempNode.InnerText);

                            tempNode = firecond.SelectSingleNode("Value");

FireConds[3].setValue(Convert.ToInt16(tempNode.InnerText));
```

```
                            break;
                        case 4:
                            tempNode = firecond.SelectSingleNode("Equation");
                            FireConds[4].setEquation(tempNode.InnerText);

                            tempNode = firecond.SelectSingleNode("Value");

FireConds[4].setValue(Convert.ToInt16(tempNode.InnerText));
                            break;
                        case 5:
                            tempNode = firecond.SelectSingleNode("Equation");
                            FireConds[5].setEquation(tempNode.InnerText);

                            tempNode = firecond.SelectSingleNode("Value");

FireConds[5].setValue(Convert.ToInt16(tempNode.InnerText));
                            break;
                        case 6:
                            tempNode = firecond.SelectSingleNode("Equation");
                            FireConds[6].setEquation(tempNode.InnerText);

                            tempNode = firecond.SelectSingleNode("Value");

FireConds[6].setValue(Convert.ToInt16(tempNode.InnerText));
                            break;
                        case 7:
                            tempNode = firecond.SelectSingleNode("Equation");
                            FireConds[7].setEquation(tempNode.InnerText);

                            tempNode = firecond.SelectSingleNode("Value");

FireConds[7].setValue(Convert.ToInt16(tempNode.InnerText));
                            break;
                    }
                }
                return;
            }

            public string getInt_Action()
            {
                return INT_Action;
            }

            public int getINT_wParam()
            {
                return INT_wParam;
            }

            public int getINT_lParam()
            {
                return INT_lParam;
            }
            public int getINT_ActionID()
            {
                return INT_ActionID;
            }

            public string getCommand()
            {
                return INT_Command;
            }

            public string getFireEquation(int FireIdent)
            {
                return FireConds[FireIdent].getEquation();
            }

            public double getFireValue(int FireIdent)
            {
                return FireConds[FireIdent].getValue();
            }

            public void setMatch(int tempValue)
            {
                Match = tempValue + Match;
                Matches = Matches + 1;
            }



            public void calcMatch()
            {
                if (Matches == Match)
                {
                    Match = Matches * 100;
                }
                else
                {
                    Match = (Match / Matches) - (7.4 * Matches);
                }
                Match = Match / Matches;
            }

            public double getMatch()
            {
                return Match;
            }

            public void clearMatch()
            {
                Match = 0;
                Matches = 0;
            }
        }

        //This class contains the methods that allows interpretations to take
place
        public class Action_Interpretor
        {
            Errors errors = new Errors();
            private ActionClass[] Actions;
            private Interpretation[] Interpretations;
            private string selectedAction, selectedCommand;
            private int selectedMatch=0, selectedActionID, wParam, lParam;


            public Action_Interpretor()
            {

                //Create Instanecs of Actions ready to be filled when sensors
operate.
                Actions = new ActionClass[8];
                for (int count = 0; count < 8; count++)
                {
                    Actions[count] = new ActionClass();
                }
```

```
    return;
}

public void addAction(int SensorID, double SensorValue)
{
    if (Actions[SensorID].getSensorValue() < SensorValue)
    {
        Actions[SensorID].setSensorValue(SensorValue);
    }

    return;
}

public double getActionSensorValue(int ActionNum)
{
    try
    {
        double temp;
        temp = Actions[ActionNum].getSensorValue();
        return temp;
    }
    catch { }
    return 0;
}



public void clearActions()
{

    Array.Clear(Actions, 0, 8);
    for (int count = 0; count < 8; count++)
    {
        Actions[count] = new ActionClass();

    }

    foreach (Interpretation inter in Interpretations)
    {
        try
        {
            inter.clearMatch();
        }
        catch { }
    }
    lParam = 0;
    wParam = 0;
    selectedAction = "";
    selectedMatch = 0;

    return;

}

public void fillInterpretations(string tempUser, string tempApp)
{
    XmlDocument docXMLFile = new XmlDocument();
    XmlNode tempData;
    XmlNodeList actionList;
```

```
                string tempName, tempCommand;
                int tempwParam, templParam;
                int count = 0, tempActionID;

                try
                {
                    docXMLFile.Load("XMLdata.xml");

                }
                catch
                {
                    errors.Error(3);
                    return;
                }
                XmlElement root = docXMLFile.DocumentElement;
                actionList = root.SelectNodes("/Config/User[./UserName='" +
tempUser + "']/App[./AppName='" + tempApp + "']/Action");
                Interpretations = new Interpretation[actionList.Count];

                foreach (XmlNode action in actionList)
                {
                    //action.
                    tempData = action.SelectSingleNode("ActionName");
                    tempName = tempData.InnerText;
                    tempData = action.SelectSingleNode("wParam");

                    tempwParam = Convert.ToInt32(tempData.InnerText,16);

                    tempData = action.SelectSingleNode("lParam");
                    templParam = Convert.ToInt32(tempData.InnerText,16);
                    tempData = action.SelectSingleNode("ActionID");
                    tempActionID = Convert.ToInt16(tempData.InnerText);

                    try
                    {
                        tempData = action.SelectSingleNode("Command");
                        tempCommand = tempData.InnerText;
                    }
                    catch { tempCommand = ""; }


                    Interpretations[count] = new Interpretation(tempUser,
tempApp, tempName, tempwParam, templParam, tempActionID, tempCommand);
                    count++;
                }
                return;
            }

            public void runInterpretor()
            {
                foreach (Interpretation interpret in Interpretations)
                {
                    for (int firecondcount = 0; firecondcount < 8;
firecondcount++)
                    {
                        try
                        {
                            if (interpret.getFireEquation(firecondcount) !=
null)
                            {
                                try
                                {
                                        if
(interpret.getFireEquation(firecondcount) == "Greater Then")
                                        {
                                            if
(Actions[firecondcount].getSensorValue() >=
interpret.getFireValue(firecondcount))
                                            {
                                                interpret.setMatch(1);


                                            }
                                            else
                                            {
                                                interpret.setMatch(0);
                                            }
                                        }
                                        else if
(interpret.getFireEquation(firecondcount) == "Less Then")
                                        {
                                            if
(Actions[firecondcount].getSensorValue() <=
interpret.getFireValue(firecondcount))
                                            {
                                                interpret.setMatch(1);

                                            }
                                            else
                                            {
                                                interpret.setMatch(0);
                                            }
                                        }
                                        else
                                        {

                                        }
                                    }
                                    catch { }
                                }
                            }
                            catch { }
                        }
                        interpret.calcMatch();
                        try
                        {
                            if (interpret.getMatch() > selectedMatch)
                            {
                                selectedAction = interpret.getInt_Action();
                                wParam = interpret.getINT_wParam();
                                lParam = interpret.getINT_lParam();
                                selectedActionID = interpret.getINT_ActionID();
                                selectedMatch =
Convert.ToInt16(interpret.getMatch());
                                selectedCommand = interpret.getCommand();
                            }
                        }
                        catch { }
                        interpret.clearMatch();
                    }

                }

                public string getAction()
                {
```

```
                return selectedAction;
            }

            public int getActionID()
            {
                return selectedActionID;
            }

            public int getwParam()
            {
                return wParam;
            }

            public int getlParam()
            {
                return lParam;
            }

            public string getCommand()
            {
                return selectedCommand;
            }

        }

        private void phidgets_FormClosing(object sender, FormClosingEventArgs
e)
        {
            ifkit.close();
            Clock.Stop();
        }
    }

    //This class handles the interactionswith the thirdparty applicaitons
    public class ThirdParty
    {
        private string className, WindowCaption;
        private System.Int32 iHandle;

        public ThirdParty(string tempClassName, string tempWindowCaption)
        {
            className = tempClassName;
            WindowCaption = tempWindowCaption;
        }

        public void sendMessage(int lParam, int wParam, int tempCommand)
        {
            iHandle = Win32.FindWindow(className, WindowCaption);
            Win32.SendMessage(iHandle, tempCommand, wParam, lParam);
        }

    }

    //This class is by the thirdparty app to make system message calls
    //This class was taken from the Microsft MSDN web page
    public class Win32
    {
        // The WM_COMMAND message is sent when the user
        // selects a command item from a menu,
        // when a control sends a notification message
        // to its parent window, or when an
        // accelerator keystroke is translated.
        public const int WM_COMMAND = 0x111;
        public const int WM_KEYPRESS = 0x102;

        // The FindWindow function retrieves a handle
        // to the top-level window whose class name
        // and window name match the specified strings.
        // This function does not search child windows.
        // This function does not perform a case-sensitive search.
        [DllImport("User32.dll")]
        public static extern int FindWindow(string strClassName,
                                          string strWindowName);

        // The FindWindowEx function retrieves
        // a handle to a window whose class name
        // and window name match the specified strings.
        // The function searches child windows, beginning
        // with the one following the specified child window.
        // This function does not perform a case-sensitive search.
        [DllImport("User32.dll")]
        public static extern int FindWindowEx(int hwndParent,
            int hwndChildAfter, string strClassName, string strWindowName);


        // The SendMessage function sends the specified message to a
        // window or windows. It calls the window procedure for the specified
        // window and does not return until the window procedure
        // has processed the message.
        [DllImport("User32.dll")]
        public static extern Int32 SendMessage(
            int hWnd,               // handle to destination window
            int Msg,                // message
            int wParam,             // first message parameter
            [MarshalAs(UnmanagedType.LPStr)] string lParam);
        // second message parameter

        [DllImport("User32.dll")]
        public static extern Int32 SendMessage(
            int hWnd,               // handle to destination window
            int Msg,                // message
            int wParam,             // first message parameter
            int lParam);            // second message parameter

        public Win32()
        {

        }

        ~Win32()
        {
        }
    }


}
```

Appendix B

XML Required for HITSI Application Prototype Demonstration

```
<?xml version="1.0" encoding="utf-8"?>
<Config>
 <User>
  <UserName>Chris</UserName>
  <App>
   <AppName>PowerPoint</AppName>
   <FileExtension>ppt</FileExtension>
   <WaitTime>1000</WaitTime>
   <ClassName>screenClass</ClassName>
   <FirstHalfWindowName>PowerPoint Slide Show - [</FirstHalfWindowName>
   <LastHalfWindowName>]</LastHalfWindowName>
   <FileName>true</FileName>
      <TimeWindow>500</TimeWindow>
   <Action>
    <ActionName>Pad One Next Slide</ActionName>
    <wParam>00010189</wParam>
    <lParam>00000000</lParam>
      <ActionID>1</ActionID>
    <FireCond>
     <SensorID>1</SensorID>
     <Equation>Greater Then</Equation>
     <Value>530</Value>
    </FireCond>
    <FireCond>
     <SensorID>0</SensorID>
     <Equation>Greater Then</Equation>
     <Value>200</Value>
    </FireCond>
   </Action>
   <Action>
    <ActionName>Pad Two Next Slide</ActionName>
    <wParam>00010189</wParam>
    <lParam>00000000</lParam>
      <ActionID>2</ActionID>
    <FireCond>
     <SensorID>3</SensorID>
     <Equation>Greater Then</Equation>
     <Value>530</Value>
    </FireCond>
    <FireCond>
     <SensorID>2</SensorID>
     <Equation>Greater Then</Equation>
     <Value>200</Value>
    </FireCond>
   </Action>
   <Action>
    <ActionName>Two People Next Slide</ActionName>
```

```
      <wParam>00010189</wParam>
      <lParam>00000000</lParam>
          <ActionID>3</ActionID>
      <FireCond>
        <SensorID>0</SensorID>
        <Equation>Greater Then</Equation>
        <Value>200</Value>
      </FireCond>
      <FireCond>
        <SensorID>1</SensorID>
        <Equation>Greater Then</Equation>
        <Value>530</Value>
      </FireCond>
      <FireCond>
        <SensorID>2</SensorID>
        <Equation>Greater Then</Equation>
        <Value>200</Value>
      </FireCond>
      <FireCond>
        <SensorID>3</SensorID>
        <Equation>Greater Then</Equation>
        <Value>530</Value>
      </FireCond>
    </Action>
  </App>
  <App>
    <AppName>PowerDVD Play</AppName>
    <FileExtension>mp3</FileExtension>
    <WaitTime>1000</WaitTime>
    <ClassName>CyberLink Video Window Class</ClassName>
    <FirstHalfWindowName>PowerDVD</FirstHalfWindowName>
    <LastHalfWindowName></LastHalfWindowName>
    <FileName>false</FileName>
    <Action>
      <ActionName>Pad One Play</ActionName>
      <wParam>0000000D</wParam>
      <lParam>011C0001</lParam>
          <ActionID>1</ActionID>
          <Command>keypress</Command>
      <FireCond>
        <SensorID>0</SensorID>
        <Equation>Greater Then</Equation>
        <Value>200</Value>
      </FireCond>
    </Action>
    <Action>
      <ActionName>Pad Two Play</ActionName>
      <wParam>0000000D</wParam>
      <lParam>011C0001</lParam>
          <ActionID>2</ActionID>
          <Command>keypress</Command>
      <FireCond>
```

```
          <SensorID>2</SensorID>
          <Equation>Greater Then</Equation>
          <Value>200</Value>
        </FireCond>
      </Action>
    </App>
    <App>
      <AppName>PowerDVD Stop</AppName>
      <FileExtension>mp3</FileExtension>
      <WaitTime>1000</WaitTime>
      <ClassName>Class of CyberLink Universal Player</ClassName>
      <FirstHalfWindowName>PowerDVD</FirstHalfWindowName>
      <LastHalfWindowName></LastHalfWindowName>
      <FileName>false</FileName>
      <Action>
        <ActionName>Stop</ActionName>
        <wParam>00000073</wParam>
        <lParam>001F0001</lParam>
            <ActionID>1</ActionID>
            <Command>keypress</Command>
        <FireCond>
          <SensorID>0</SensorID>
          <Equation>Less Then</Equation>
          <Value>200</Value>
        </FireCond>
        <FireCond>
          <SensorID>2</SensorID>
          <Equation>Less Then</Equation>
          <Value>200</Value>
        </FireCond>
      </Action>
    </App>
  </User>
</Config>
```