International Journal of Semantic Computing

World Scientific
www.worldscientific.com

ASSIGNING SEMANTICS TO SENSED HUMAN ACTIONS: A FRAMEWORK AND STEPS TOWARDS AN ABSTRACT MODEL

SENG W. LOKE, CHRIS TIVENDALE

Department of Computer Science and Computer Engineering, La Trobe University
Melbourne, Bundoora, VIC 3086, Australia
s.loke@latrobe.edu.au
http://homepage.cs.latrobe.edu.au/sloke

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

While there have been many applications that rely on sensors for human computer interaction, for monitoring environments, and for smart context-aware applications, many systems are built in an ad hoc manner, and targets specific domains. There will be a growing demand for such applications whether for everyday life or games. This paper reports on work in two directions towards general development strategies for sensor-driven systems: one is an XML programmable framework and the other an abstract relational model for specifying such sensor-based human computer interaction systems (together with the ability to formally define properties of such systems). While we describe a specific implementation and model, the paper advocates high-level programmability, and development via formal abstract specifications, as two important areas of research towards systematic development of sensor-based interaction systems.

Keywords: sensor interpretation; software engineering for sensor-based interaction.

1. Introduction

There has been much work on multimodal input (e.g., using gestures, eye-tracking, speech and sounds, movement, touch, pen, keyboard, device tilt and orientation, etc) and output, and sensor-driven interfaces (e.g. [8,9]). Indeed, the future generation of interaction seems to be heading in this direction. Such work demonstrates the ability to use multiple inputs together to interact with a computer application. Toolkits such as Phidgets* have enabled sensor-driven applications to be developed conveniently, without the need to build hardware. For the software development aspects, this paper proposes HITSI (Human Interaction Through Sensor Interpretation),

(i) a framework for human interaction with computer applications through *interpreting concurrent inputs from multiple sensors*, a process termed *sensor interpretation* (which essentially assigns semantics to the sensor inputs or, indirectly, to the human actions detected by the sensor inputs, where semantics is viewed operationally as yielding some system action to be taken), and
(ii) a generalization and abstraction of HITSI.

The two main contributions of this paper are:

* http://www.phidgets.com

(i). a software framework for human interaction via sensor interpretation which can be XML-programmed to create different applications using different sensor inputs; and,
(ii). a conceptualization, formalization, and generalization of sensor interpretation, given our assumptions of concurrent sensor inputs, in terms of an abstract model of relations. We present (i) and then (ii) to first provide a concrete grounding for the model in (ii).

This paper is organized as follows. Section 2 describes our prototype HITSI system. Section 3 abstracts from the HITSI system to an abstract model of HITSI based applications, presenting a list of properties of these systems and compositional operators. Section 4 discusses related work and Section 5 concludes.

2. The HITSI System

The HITSI system takes multiple sensor inputs and reads in an XML document containing rules about what actions to map sensor inputs to. It has been prototyped using the Phidgets toolkit though adaptors can be build to the system for any sensor (wireless or wired).

2.1. *Sensor Interpretations*

A sensor interpretation requires the mapping of sensor inputs to a particular *concept*, representing the semantics and operational effects of a sensor interpretation. A group of sensor readings are first read into the system and the information contained in these readings is extracted and temporarily stored. The system looks at all the sensor readings and considers each collection of readings within a time window as belonging to a group and attempts to match them with a concept. A concept is operationalized as a two-part rule: (i) a set of Fire Conditions (antecedent of rules described later) and (ii) what action to take if the fire conditions are met. The actions are predetermined in an XML file and are available to be looked up by the system. A diagram of this process is detailed in Figure 1 below. A Fire Condition can be compared to sensor inputs to give a true or false value. Every fire condition essentially consists of two parts. The first is the identification of which the sensor whose input this fire condition is to be compared against. The second part of the fire condition is what needs to be satisfied (e.g., a Boolean expression on the sensor value) to give the fire condition a true value.

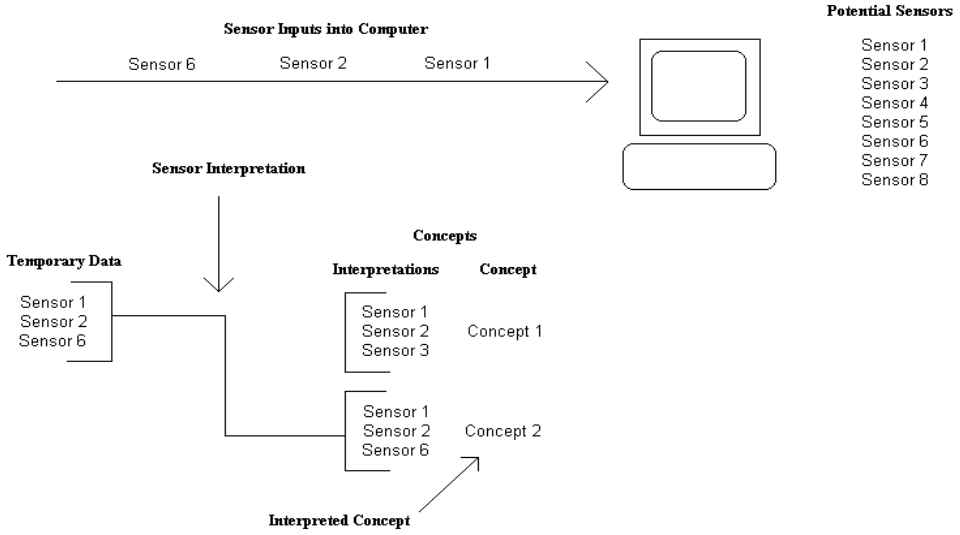


Fig. 1. Sensor Interpretation

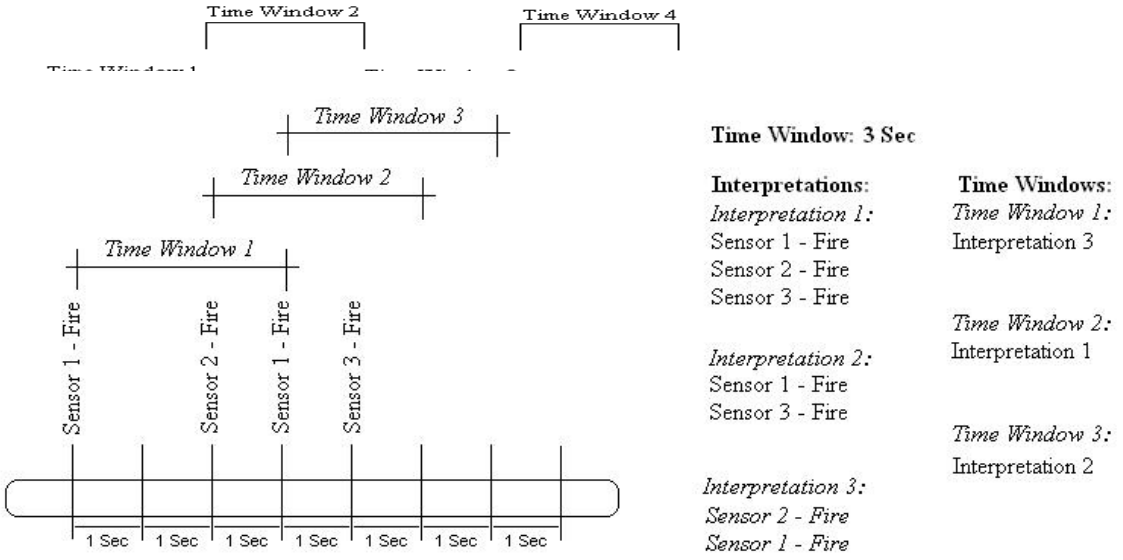
An example of where an interpretation is used could be a room equipped with a motion sensor, light sensor and pressure sensor on the floor in front of a projector. When the light sensor is reading a "light" room along with movement recorded in the motion sensor but no pressure recorded on the pressure 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 concepts in these examples are a) "pre lecture" and b) "lecture is underway".

When attempting to interpret sensor inputs from multiple sensors, there could be multiple sensor interpretations and so, no unique concept assignment. An example is if a room with three sensors (one, two and three) has an interpretation which is a combination of readings from sensor one and two, while another interpretation is a combination of sensor two and three. Which interpretation should be used if all of these are possible? More sophisticated reasoning can be applied, but for simplicity in this prototype, in the event of such ambiguity, a procedure needs to be developed such that an interpretation can be selected. Implementing unique "priorities" for interpretations would be a way to resolve these conflicts. By introducing a priority to interpretations, the ambiguity in the above situation disappears as the higher priority interpretation is selected.

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 (even if *almost* simultaneousy); therefore, an application using human input through multiple sensors needs to allow a time window for these human actions and sensor readings to be considered as concurrent and to be processed by the computer application. Figure 2 shows an example of multiple sensor inputs occurring over a period of time.

Fig. 2. Multiple Sensor Inputs Over Time

It would be very difficult to attempt to interpret these sensor inputs as they arrive 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. Figure 3 shows how the data presented in the previous example can be grouped into smaller time groups. These sections of time are the time windows.



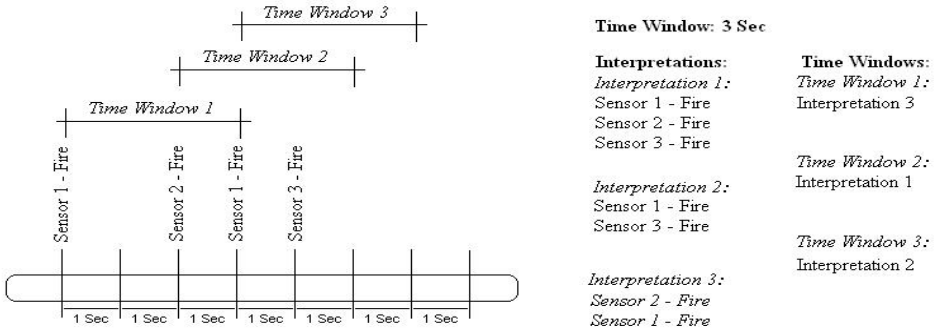


Fig. 4. Sliding Time Window

To overcome the problems, we simply use the non-overlapping time windows as in Figure 3, which avoids ambiguities that arise between interpretations as sensor inputs are deemed to occur within one distinct time window.

2.2. *Coupling with External Applications*

One of the requirements of the HITSI Application is the ability to control actions in a third party software application. We take advantage of the Windows system messages to send information to third party applications These system messages are standard to all windows based applications and could therefore be understood and used by all third party windows based applications.

2.3. *Architectural Overview and Prototype Implementation*

Figure 5 shows the way in which the information flows in the HITSI prototype. Sensors take readings and forward these onto the Phidgets Interface Control Board. From here sensor inputs are passed to the HITSI system. The HITSI system monitors sensor inputs and stores them for future interpretation. By comparing sensor inputs to stored interpretation definitions, an interpretation is chosen and a system message is sent to a third party application. Such a mapping to actions can be programmed via an XML file described later.

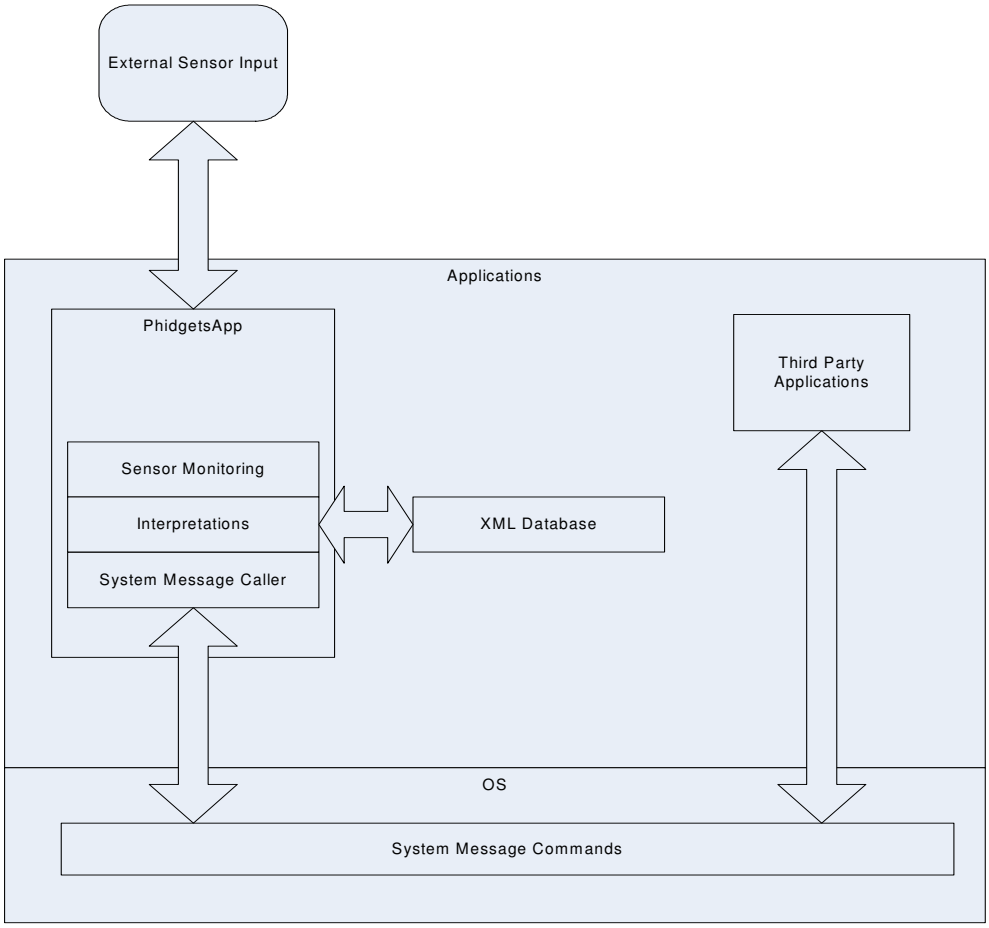


Fig. 5. Architectural Overview

The flow chart in Figure 6 gives the possible steps that the HITSI system experiences during operation.

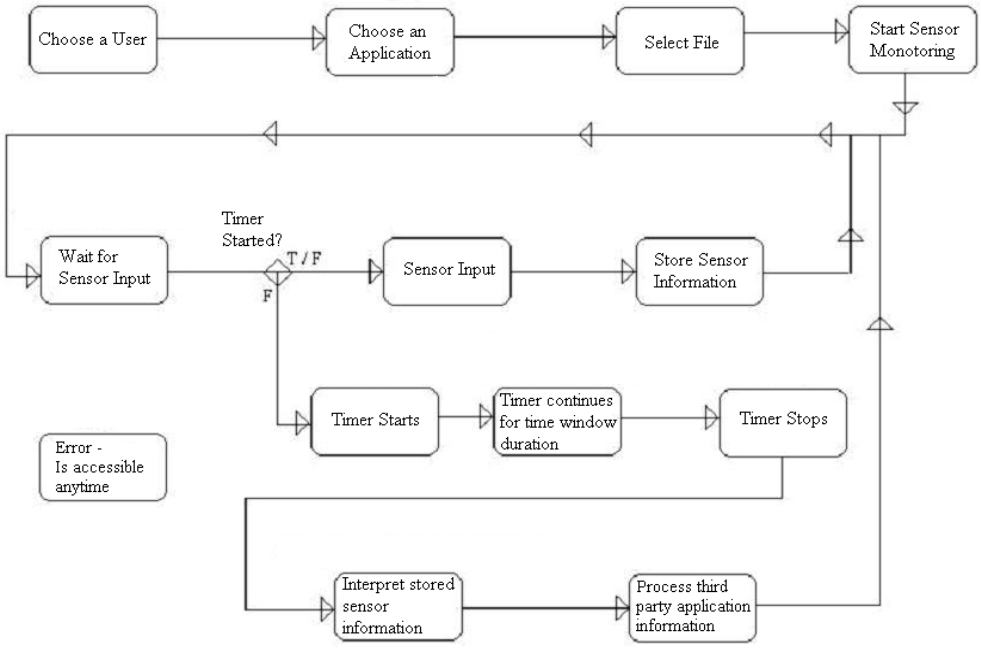


Fig. 6. Flow Chart

Initially, a user must be identified. 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 (e.g., a .ppt file) in which they will control through the third party application (e.g., Microsoft PowerPoint). 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 for interpretations. The HITSI system then waits 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 timer is activated and HITSI system continues to wait for sensor inputs. When a predetermined time has elapsed since the sensor input (i.e., end of a time window), 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 wait for the next sensor input. Timer is restarted for each time

window. The system reads behaviour rules in an XML format storing it in an XML database. Figure 7 shows the main classes of the HITSI prototype. The UI classes (UI Main and UI Sensor Interpreter) implements the UI for HITSI; the UI to the sensor interpreter enables testing by direct input of fire conditions. 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 possible error messages. 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 time window and are removed after being involved in an interpretation. All possible interpretations are stored in the Interpretation class and each fire condition for these interpretations is stored in the FireCond Class. After an interpretation is selected the Third Party class is used to interact with a third party application.

The following is an example XML rules set in an XMLData.xml file. This example contains one user, one application and one action but there could be more.

```
<Config>
     <User>
    <UserName>Chris</UserName><WaitTime>1000</WaitTime>
    <App>
      <AppName>PowerPoint</AppName>
      <FileExtension>ppt</FileExtension>
      <ClassName>screenClass</ClassName>
             <FirstHalfWindowName>PowerPoint[</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 Than</Equation>
          <Value>200</Value>
        </FireCond>
         <FireCond>
          <SensorID>1</SensorID>
          <Equation>Greater Than</Equation>
          <Value>500</Value>
        </FireCond>
        <FireCond>
          <SensorID>2</SensorID>
          <Equation>Less Than</Equation>
          <Value>100</Value>
        </FireCond>
        <FireCond>
          <SensorID>3</SensorID>
          <Equation>Greater Than</Equation>
          <Value>300</Value>
        </FireCond>
      </Action>
    </App>
  </User>
</Config>
```

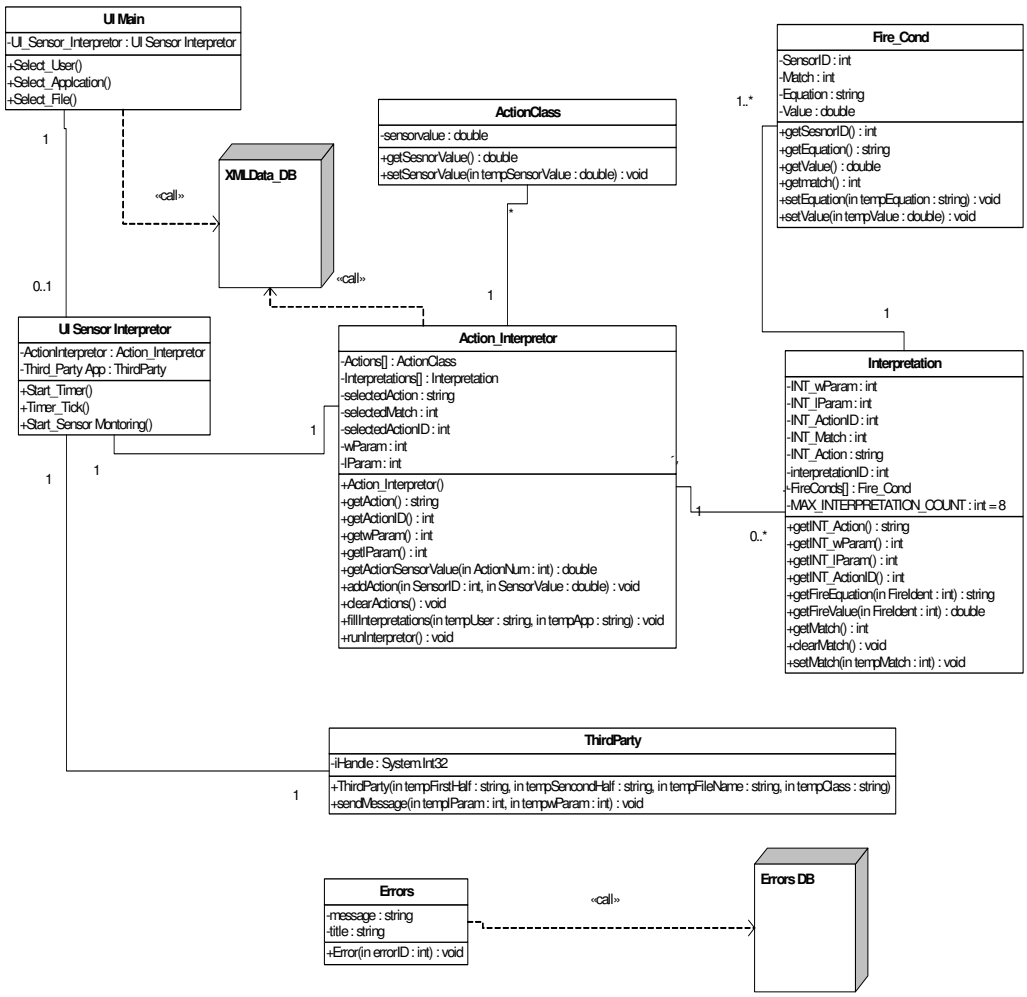


Fig. 7. Main Classes of the HITSI prototype

The configuration in the above XML document is explained as follows:

1. one user (of username Chris) and a time window of one second
2. one application called "PowerPoint" with file extension ".ppt"
3. the PowerPoint third party application is identified by the Classname "screenClass" and the window name "PowerPoint [ExampleFile.ppt]"
4. 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
5. the fire conditions for the "next slide" action are: sensor 0 must be greater than 200, sensor 1 must be greater than 500, sensor 2 must be less than 100, sensor 3 must be greater than 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 the third party application. One could configure HITSI so that all the fire conditions (such as for "next slide" above) must be met before the action is carried out, but more generally, HITSI can also use a mix of priorities and a scoring system, i.e. each fire condition can be given a priority value, and a threshold score for executing the action can be pre-defined. A score is computed given the sensor readings. For example, given the action with the four fire conditions above, and a set of readings, we compute the score as follows:

`score(readings) = w1*s1 + w2*s2 + w3*s3 + w4*s4`

where `si=1` if the fire condition `i` is satisfied and `si=0` otherwise, and the `wi` values correspond to the relative priorities of the condition (they sum to `1`). The action is carried out if `score(readings) > action_threshold`.

Anecdotal experimental evidence shows that a threshold of 75% of the maximum possible score of a reading proves to be useful for accommodating a small margin of error when users attempt sensor inputs with the Phidgets sensors set up we used. In the specific case, 75% means that, given equal priorities, an interpretation with at least three satisfied fire conditions need to be met or, if unequal priorities, then sufficient higher priority conditions need to be met.

The implementation of the HITSI application makes use of Microsoft Windows WM_COMMAND messages and the WIN32.SendMessage function in C#. WM_COMMAND messages are issued by the HITSI application to the OS. 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.

2.4. *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. With some experimentation, we determined that 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, though adequate sensor inputs for most applications (e.g., Powerpoint, etc).
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 provide multiple sensor inputs.

Another performance issue that can restrict the time window to a minimum of half a second is the time it takes for the application to interpret all the sensor readings and for the third party application to perform the requested action. Timing results (on a standard modern desktop computer) show that the time duration from just before the interpretation function was called, after the interpretation had been decided, to the command being sent to the third party application was 0.1 secs. This time delay is virtually imperceptible by humans. (The number of interpretations in this timing was 249, which is many more then would be used in normal operation).

2.5. *Accuracy*

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 (which was experimentally determined) 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 subjected to a set of criteria that attempts to understand what the user wished to achieve. However, we note that the use of 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.

2.6. *User Programmability*

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, and (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 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, though graphical based tools can be further built for users.

2.7. *An Example Application for Illustration*

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 (where visitors can come in and drive the presentation via sensors). The HITSI Application would be set up in a regular room with the Phidgets Interface Kit including two motion sensors and two force sensors. The layout of the room would appear as in Figure 8.

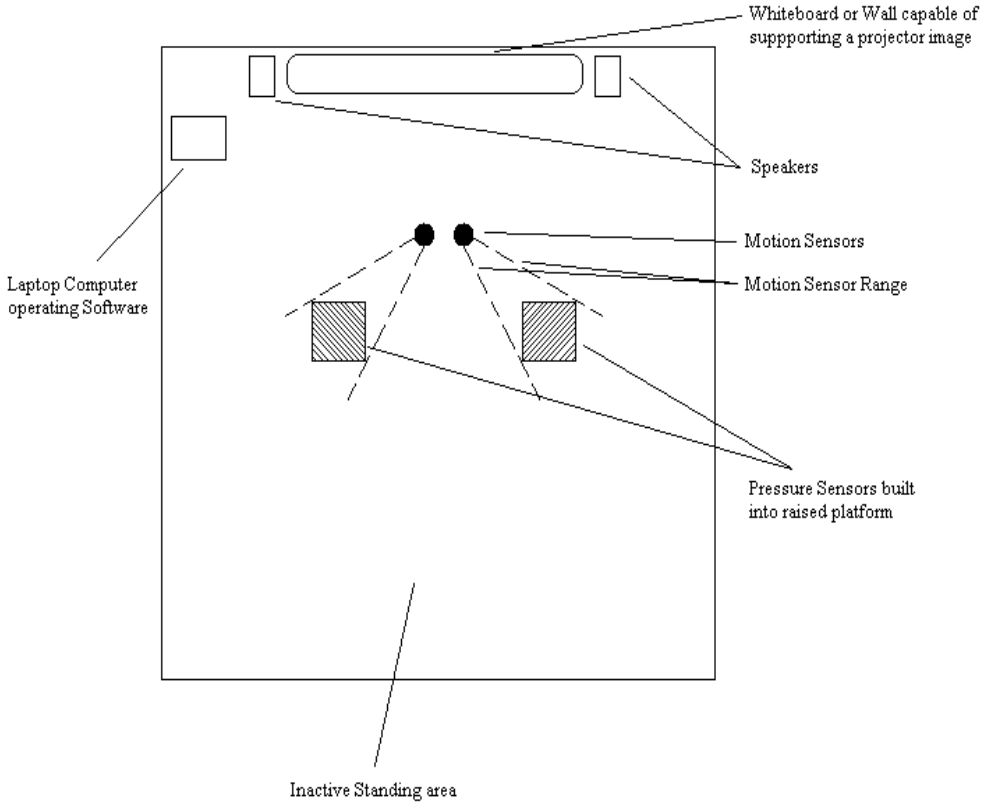


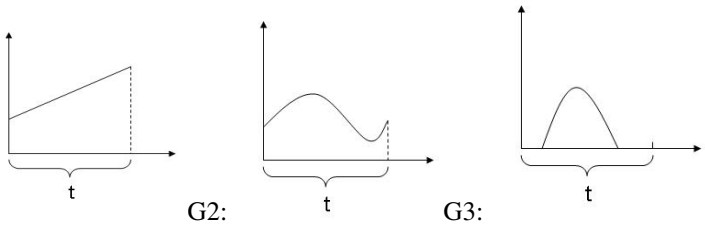
Fig. 8. HITSI Demonstration Setup

Prospective students or visitors enter the room. Whilst nobody is standing on the raised wooden platforms (with force sensors underneath) 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 La Trobe University. 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.

3. Generalizing - The HITSI Abstract Model

We consider an abstract model of a system which takes sensor inputs over predefined time windows, interprets the inputs in each time window, and maps the sensor interpretations to actions. The model views such a system abstractly as simply a relation between sensor inputs and actions. For simplicity, we assume in the following that each sensor returns a value within a given range (e.g., 1 to 100). Letting **S** be a set of possible sensor readings (the exact nature of which we do not consider below), and **A** be a set of actions (e.g., starting or stopping a Windows application, and switching the lights on), such a system **H** can be viewed as representing a binary relation between a set (or bag) of sensor readings and a set of actions,† i.e., we write
$\mathbf{H} \subseteq \wp(\mathbf{S}) \times \wp(\mathbf{A})$,‡ and if $(s, a) \in \mathbf{H}$, then $\mathbf{s} \subseteq \mathbf{S}$ and $a \subseteq \mathbf{A}$.
The set of sensor readings corresponds to readings obtained within a given (small) time window, and are viewed by **H** as being concurrent (though they may not be so). For example, suppose s={100,200,200} corresponding to readings from a motion sensor, a touch sensor and a force sensor, respectively, assuming that the time window is small enough so that each reading remains effectively constant throughout the time window. If the time window **t** was sufficiently large, s is no longer a set of discrete values but a set of graphs (each graph represented by a set of readings over period **t,** the exact number of points depending on the sampling rate with respect to **t**), i.e., s={G1,G2,G3}, where, for example:



G1: G2: G3:

In this case, we have effectively a "signature" of readings from the three sensors. Such a "signature" might then be interpreted, i.e. map to some action (e.g., such advancing a PowerPoint slide). One could also "summarize" the readings over the time period, e.g., computing an average value for the readings in each of G1, G2 and G3, and then map the

† An action can be an operation on an application (e.g., play or stop music in Windows Media Player), or a Web service invocation to control devices including a lamp, drapes, etc [10]

‡ We can also define $\mathbf{H} \subseteq \wp(\mathbf{S}) \times \mathbf{A}$, but more general is $\mathbf{H} \subseteq \wp(\mathbf{S}) \times \wp(\mathbf{A})$.

set of averaged readings to a set of actions. The discussion which follows will apply, independently of these details.

In the case of readings being taken over a period T=k**t**, for some integer k, the system partitions the readings over T into k segments, and interprets each segment individually, assigning a set of actions to each segment (i.e., a set of actions for the set of sensor readings in each segment). For example, over a time T=3**t**, there are three time segments (say, **t**1, **t**2 and **t**3, one preceding the other), then using **H** maps the sensor readings in each time segment to a set of actions, i.e. sensor readings in segment **t**1 is mapped to a set of actions a1, readings in **t**2 to a set of actions a2, and **t**3 to a3. The three sets of actions a1, a2, and a3 should then be scheduled to be performed in sequence, within their respective time windows.

Our approach implies that a relation $\mathbf{H} \subseteq \wp(\mathbf{S}) \times \wp(\mathbf{A})$ can act as an abstract specification of a particular HITSI system, providing us the mechanism to define precisely a vocabulary of concepts that we can use to talk about these systems. We might use these ideas informally but we aim to then develop "HITSI genre" systems based on these abstract (and algebraic - as inspired by [7]) specifications. We consider three categories of formalization based on this abstract model: system properties, inter-system relationships and composition operators.

3.1. *System Properties*

We can define properties of such a system in terms of properties of its corresponding relation. For illustration, and not being exhaustive, we consider here four properties: determinism, strictness, inverse-completeness, and continuity.

- **Determinism.** Given an $s \in \wp(\mathbf{S})$, whenever we have (s,a1) $\in$ **H** and (s,a2) $\in$ **H,** such that a1 $\neq$ a2. If so, the system (or its corresponding relation) is *non-deterministic*; otherwise, it is *deterministic*.
- **Strictness.** Given that a HITSI system has a fixed number of sensors, in an interaction over a time window, it might be that some sensors detect nothing – we represent this by assuming that every sensor has a default undefined reading to start with (denoted by "$\perp$"). Should the system then treat $\perp$ readings as "any value" or "no value"? Consider a system with four sensors: a motion sensor m, a touch sensor t, a force sensor f, and a vibration sensor v. An input s={m:40,t:60,f:90,v:$\perp$} during a time window could mean that the vibration sensor was not affected at all. Given a system with n sensors s_1 to s_n, i.e. inputs to the system are of the form:

  $\{s_1:v_1, \ldots, s_n:v_n\}$

  where each v_i is either the default "$\perp$" or some other valid sensor reading, for each possible input s, we can define a set R(s) of sets of readings, where each set of reading has values from s or is undefined, i.e.

  $R(s)=\{\{s'_1:v'_1,\ldots,s'_n:v'_n\} \mid \forall i,\ s'_i:v'_i \in s \text{ or } v'_i=\perp\}$

  Then, we say that the system **H** is *strict* iff

  $\forall$ (s,a) $\in$ **H**, (s',a) $\notin$ **H** for all s' $\in$ R(s)

i.e. a strict system **H** will produce an action set "a" only when (or strictly when) all its required sensor readings are detected; otherwise, the system is *non-strict*. Note that one can also define strictness with respect to particular actions.

- **Inverse-Completeness.** In what sense can we say that the system is complete? We provide a definition of completeness that takes into account a simple algebraic structure on the set of possible actions. Suppose, the set of possible actions **A** forms a group (in the group-theoretic sense)[§] with respect to a composition operator (simply, the sequential operator, denoted by ";", say), then, there is an identity action $e \in \mathbf{A}$, and for each action $\alpha \in \mathbf{A}$, there is an inverse action of α, denoted by $\alpha^{-1} \in \mathbf{A}$, such that $\alpha; \alpha^{-1} = \alpha^{-1}; \alpha = e$

  Ignoring changes of state or side-effects at this time, one can think of an action α, say, as "go to next slide" in a MS PowerPoint presentation, and α^{-1} as "go to previous slide", or "turn left" and "turn right" of a car. A system **H** is said to be inverse-complete with respect to a group of actions, if for each $(s,\{\alpha\}) \in \mathbf{H}$, there exists $(s',\{\alpha^{-1}\}) \in \mathbf{H}$, i.e. for any action (within the group) that one does, there is some way to perform the inverse action of the action.

So far, **H** considers only multi-sensor inputs falling within the same time window. We can enrich **H** to map multi-sensor inputs across different time windows to actions by associating a time window stamp (denoted by `t1`, `t2`, `t3` … $\in \mathbf{T}$, the set of time window stamps) with each set of sensor readings. So,

$\mathbf{H} \subseteq (\wp(\wp(\mathbf{S}) \times \mathbf{T})) \times \wp(\mathbf{A})$

(In defining such a **H**, we may want to ignore certain readings in a time window, and this can done by using the value $\perp$ for the sensor reading in that time window.). So, with a system of `n` sensors,

$(\{(\{s_1:v_1, \ldots, s_n:v_n\}, \mathbf{t}1), (\{s_1:v'_1, \ldots, s_n:v'_n\}, \mathbf{t}2),$
$(\{s_1:v''_1, \ldots, s_n:v''_n\}, \mathbf{t}3), (\{s_1:v'''_1, \ldots, s_n:v'''_n\}, \mathbf{t}5)\},$
$\{a1,a2\}) \in \mathbf{H}$

means that the collection of sensor readings with values $\{s_1:v_1, \ldots, s_n:v_n\}$ in time window $\mathbf{t}1$, $\{s_1:v'_1, \ldots, s_n:v'_n\}$ in time window $\mathbf{t}2$, $\{s_1:v''_1, \ldots, s_n:v''_n\}$ in time window $\mathbf{t}3$, and $\{s_1:v'''_1, \ldots, s_n:v'''_n\}$ in time window $\mathbf{t}5$, will be interpreted together and mapped to actions a1 and a2.

- **Inverse-Continuity.** Given a *complete* system, we might want to know if the inverse can be initiated immediately after an action, i.e. formally, given a time window of a certain size, suppose for some time window `t(i-1)`, we have

  $(\{(\{s_1:v_1, \ldots, s_n:v_n\}, t(i-1))\}, \{\alpha\}) \in \mathbf{H}$

  Then, in the current time window, `ti`, does there exists values v'_i such that

  $(\{(\{s_1:v'_1, \ldots, s_n:v'_n\}, ti)\}, \{\alpha^{-1}\}) \in \mathbf{H}$?

  which asks whether it is possible to provide some sensor inputs in the current time window to initiate the inverse of the action initiated in the previous time window. Note that if this is not possible, it could be due to a number of issues, e.g., simply the

§ The actions may be that of some application (e.g., a set of game operations, operations on an appliance, etc), where an algebraic structure is present.

way the system was designed, it is not possible to perform the inverse action fast enough. So, suppose the answer to the above question is no, then the system corresponding to or realizing **H** is said to be inverse-discontinuous with respect to α, and inverse-continuous with respect to α otherwise. What the notion of continuity attempts to capture is whether the system allows "undo" or reversal of actions fast enough.

The above four properties attempt to characterize a system formally. While these properties are general and not application-specific, further properties can be defined which are tailored to a particular application (e.g., a game or a smart home application), but as we demonstrated, specifiable in terms of relations.

3.2. *Inter-System Relationships*

We can define various relationships between systems as relationships between relations that represent their behaviour. We define the following relationships:

- **Equality.** Two systems **H** and **H'** are equal if their corresponding specification relations are equal, i.e. **H=H'**. For the same sensor inputs, both systems map to exactly the same actions. Note that this holds even if the internals of both systems are entirely different – this is a black-box equality.
- **Commonality.** $\mathbf{H} \cap \mathbf{H'} \neq \varnothing$ which means that for some sensor inputs, both systems will provide exactly the same actions. The two systems then have *commonality*.
- **Monotonic extension.** A system **H'** is a *monotonic extension* of **H** iff $\mathbf{H} \subseteq \mathbf{H'}$, i.e. if we started with a system such that $(s,a) \in \mathbf{H}$ for some s and a, then if we extended the system, we might want to ensure that what was working before still works. This definition easily extends to the kind of extensions where more sensors have been added, say from n sensors to $n+m$ sensors: $(s,a) \in \mathbf{H}$ implies $(s',a) \in \mathbf{H'}$, where $s'=s \cup \{s_{n+1}: \perp, \ldots, s_{n+m}: \perp\}$, and **H'** necessarily non-strict. Note that this relationship between two systems can apply even if the two systems have been build by different people and are internally different.

3.3. *System Composition*

Given two systems (or their relational specifications), we can compose them to obtain a combined system specification whose behaviour is dependent on the individual system behaviours in a certain way, as defined by the semantics of the composition operator. We provide the following composition operators:

1. **Union.** Given two systems, **H** and **H'**, their (set-theoretic) union, denoted by "$\cup$" is given by: $\mathbf{H} \cup \mathbf{H'} = \{(s,a) \mid (s,a) \in \mathbf{H} \text{ or } (s,a) \in \mathbf{H'}\}$
   Note if both $(s,a) \in \mathbf{H}$ and $(s,a) \in \mathbf{H'}$, the union composition is non-deterministic.
2. **Action-union**. Given two systems, **H** and **H'**, their *action-union*, denoted by"$\cup_{au}$" is given by: $\mathbf{H} \cup_{au} \mathbf{H'} = \{(s, a \cup a') \mid (s,a) \in \mathbf{H} \text{ and } (s,a') \in \mathbf{H'}\}$
   which means that given sensor inputs, actions from both **H** and **H'** are initiated. Sometimes, there could be conflicts since "s" can lead to action set containing α and its inverse α^{-1} in which case they either cancel each other out and neither is

performed or one is carried out and later the other. Note that this operation leaves out actions initiated by inputs that did not initiate actions in both systems.

3. **Intersection**. Given two systems, **H** and **H'**, their (set-theoretic) intersection, denoted by "$\cap$" is given by: $\mathbf{H} \cap \mathbf{H'} = \{(s,a) \mid (s,a) \in \mathbf{H} \text{ and } (s,a) \in \mathbf{H'}\}$
4. **Action-intersection**. Given two systems, **H** and **H'**, their *action-intersection*, denoted by "$\cap_{ai}$" is given by: $\mathbf{H} \cap_{ai} \mathbf{H'} = \{(s, a \cap a') \mid (s,a) \in \mathbf{H} \text{ and } (s,a') \in \mathbf{H'}\}$, which means that for a set of sensor inputs, only actions in common from both **H** and **H'** are initiated.
5. **Sensor-intersection**. Given two systems, **H** and **H'**, their *sensor-intersection*, denoted by "$\cap_{si}$" is given by: $\mathbf{H} \cap_{si} \mathbf{H'} = \{(s \cup s', a) \mid (s,a) \in \mathbf{H} \text{ and } (s',a) \in \mathbf{H'}\}$ Note that we union the sensor inputs,** i.e. combine the conditions for the action set a: the input has to be of particular values on the sensors of both systems (within the same time window) before an action set can be initiated – we assume that the set of sensors on both systems are disjoint, since typically each system would have its own set of sensors.
6. **Parallel-union**. Note that many other operators can be defined such as the following which incorporates sensor-intersection with action-union, which we call parallel-union (denoted by "$\|_u$"). The meaning of this operator is to combine sensor inputs and to combine resulting actions but in such a way as not to interfere with each other:

   $\mathbf{H} \|_u \mathbf{H'} = \{(s \cup s', a \cup a') \mid (s,a) \in \mathbf{H} \text{ and } (s',a') \in \mathbf{H'}\}$

And a corresponding **Parallel-intersection:**

$\mathbf{H} \|_i \mathbf{H'} = \{(s \cup s', a \cap a') \mid (s,a) \in \mathbf{H} \text{ and } (s',a') \in \mathbf{H'}\}$

Note that in the above, we have assumed that
$\mathbf{H} \subseteq \wp(\mathbf{S}) \times \wp(\mathbf{A})$, and $\mathbf{H'} \subseteq \wp(\mathbf{S'}) \times \wp(\mathbf{A'})$ (i.e., $\mathbf{H}, \mathbf{H'} \subseteq \wp(\mathbf{S} \cup \mathbf{S'}) \times \wp(\mathbf{A} \cup \mathbf{A'})$) so that each composition $\mathbf{H} \circ \mathbf{H'} \subseteq \wp(\mathbf{S} \cup \mathbf{S'}) \times \wp(\mathbf{A} \cup \mathbf{A'})$
where $\circ \in \{\cup, \cup_a, \cap, \cap_s, \|_u, \|_i\}$, and we assumed that **S** and **S'** are disjoint (i.e., the two systems don't share sensors) but not for **A** and **A'** (they can have common resulting actions).

7. **Restriction.** Given two systems, **H** is restricted by **H'**, denoted by "\", is given by:
   $\mathbf{H} \setminus \mathbf{H'} = \{(s,a) \mid (s,a) \in \mathbf{H} \text{ and } (s,a) \notin \mathbf{H'}\}$, which means that given a sensor input s, we initiate a set of actions a from **H** provided that the exact same set of actions are not also initiated by **H'**.

   However, we can achieve a finer granularity of control since "a" is a set of actions, as follows which we call action-restriction.
8. **Action-restriction**. Given two systems, **H** is action-restricted by **H'**, denoted by "$\setminus_{ar}$", is given by: $\mathbf{H} \setminus_{ar} \mathbf{H'} = \{(s, a \setminus a') \mid (s,a) \in \mathbf{H} \text{ and } (s,a') \in \mathbf{H'}\}$, which means given sensor inputs, actions are initiated from **H** which are not also initiated by **H'**.
9. **Sensor-restriction.** Given two systems, **H** is sensor-restricted by **H'**, denoted by "$\setminus_{sr}$", is given by: $\mathbf{H} \setminus_{sr} \mathbf{H'} = \{(s,a) \mid (s,a) \in \mathbf{H} \text{ and } \forall a' \neq \emptyset, (s,a') \notin \mathbf{H'}\}$, which means that we only take mappings from **H** which does not map to any non-empty action set in **H'**. So if a sensor input s maps to some actions by **H**, but also to some non-empty action set by **H'**, the system $\mathbf{H} \setminus_{sr} \mathbf{H'}$ will not take any action.

** Note that we call this a form of intersection since the intuition is that union of sensor inputs places additional conditions for actions, and so, is more constraining.

In general, given a sensor input s to a system **H**, how do we model the fact that **H** does not take any action for s? One way is $(s,a) \notin \mathbf{H}$ for any action set a, i.e. we say that **H** is undefined for the input s. Another way is to redefine the relations such that there is always a mapping for any input, i.e., **H** is always defined for any input, but just that for some inputs, **H** maps to an empty action set; we have that for any sensor input s, either $(s, \emptyset) \in \mathbf{H}$ or $(s, a) \in \mathbf{H}$, for some $a \neq \emptyset$. Either option is possible.

Given a system **H**, we might want to create a new system **H''** which extends **H** by adding the mappings for sensor inputs from another system **H'** as long as there is no mapping in **H** for those sensor inputs. We define a new operator as follows to represent **H''** as a composition of **H** and **H'**:

10. **Overriding-union** (denoted by "$\lhd$"), defined by $\mathbf{H''} = \mathbf{H} \lhd \mathbf{H'} = \mathbf{H} \cup (\mathbf{H'} \setminus_{sr} \mathbf{H})$
    Certainly, **H''** is a monotonic extension of **H**. Note that another interpretation of overriding-union is an inheritance operator analogus to object-oriented inheritance of a class from its superclass.

Hence, we have a rich set of operators which can be used to compose new systems (or their specifications) from existing systems. The set of operators are non-exhaustive. At the specification level, one should be able to define simpler systems and then compose them to form more complex systems, with reuse. For example, we might have a system which opens a door if it detects particular user and which switches on the light for another user, and then attempt to compose this so that light and door is operated upon for both users. While we do not discuss this here, we can examine the algebraic properties to determine whether properties are preserved under a given composition.

Scenario Example. In the home, one (i) may step into the living room causing a lamp to come on and (ii) can sit on a sofa and tap its side to switch the TV on; both (i) and (ii) are orthogonal, and in fact may be due to two different systems, say P and Q respectively. In abstract terms, we have a union of two systems (or relations), one mapping steps to turning the lamp on and the other mapping actions on the sofa to turning the TV on. In general, a smart home may contain many such mappings yielding, in our terminology, a union of many (abstract) systems. Suppose another system R is added which causes the curtains to be drawn when one steps into the living room. Then a combined system that maps stepping into the living room to turning the lamp on and drawing the curtains is abstractly the action-union of the two systems P and R. With HITSI, the union of two XML rule-sets corresponds to a union of two abstract systems. The purpose of these composition operators is (i) compositional specification, where one can specify a system in terms of simpler specifications, and (ii) if the composed specifications can be taken and translated into real executables, one can assemble different systems based on a core set of basic systems (future work on HITSI is to be extended to realize this).

In the multiuser case, given a set of users (or their IDs) **U,** each sensor reading might be attributed to a user, and the system is now a relation:
$\mathbf{H} \subseteq \wp(\mathbf{U} \times \mathbf{S}) \times \wp(\mathbf{A})$. Each $(\{(u_1,s_1),\ldots,(u_n,s_n)\},a) \in \mathbf{H}$, where $u_i \in \mathbf{U}$, maps sensory inputs from multiple users to some set of actions a. Even though we have mapped to a set of actions, this set can be empty or contain only one action.

4. Related Work

Since their inception, there has been much work using Phidgets not only for tangible inputs to computer applications[††] but also for physically tangible outputs [1,2,3,4,5]. The notions of Tangible User Interfaces and Pervasive Gaming rely on sensors for their realization. Games have also been developed in [6,1]. However, we see that the mapping between sensor actions and operations on computer applications have not been comprehensively discussed in the literature as we do here. Also, we presented a mathematically abstraction for the class of sensor-driven applications with discrete (or discretized) valued sensor inputs. Most of the work on tangible interfaces to date are practical and implementation based rather than a formalization as we do here.

Note, our work here deals with mapping a set of inputs to actions immediately - data mining a history of inputs or learning (over time) behaviours of users are not within the scope of this paper. Other work on knowledge-based situation modeling exists (e.g., [11]) but are not related to the notion of situations described here.

5. Conclusion and Future Work

The paper has argued for a general and more systematic approach to building the class of sensor-driven applications, which we envision will be in growing demand. To illustrate what we mean, we described (i) a software framework for human interaction via sensor interpretation which can be XML-programmed to create different applications which might use different combinations of sensor inputs; and, (ii) a conceptualization and formalization of sensor interpretation, given our assumptions of concurrent sensor inputs, in terms of an abstract model of relations. The latter serves as steps towards specifications of systems which can be reasoned about, composed and from which actual systems can be systematically derived - the partial generation of such systems from specifications is what we intend to explore in the future. Each XML HITSI program can be considered a relation (each HITSI system with its fixed set of sensors and attached applications can be considered as a placeholder for realizing a set of relations, and multiple instances of the HITSI system can be used to realize compositions of relations).

References

[1] Jung, B., Schrader, A. and Carlson, D. Tangible Interfaces for Pervasive Gaming. Proceedings of Digital Games Research Association International Conference (DIGRA), 2005.

[2] Kimura, H., Tokunaga, E., Okuda, Y. and Nakajima, T. CookieFlavors: Easy Building Blocks for Wireless Tangible Input. Proceedings of the Conference on Human Factors in Computing Systems (CHI), ACM Press, pp. 965 – 970, 2006.

[3] Klemmer, S.R., Li, J., Lin, J., and Landay, J.A. Papier-Mâché: Toolkit Support for Tangible Input. CHI Letters, Human Factors in Computing Systems 6(1), 2004.

†† http://grouplab.cpsc.ucalgary.ca/phidgets/

[4] Koleva, B., Benford, S., Ng, K. and Rodden,T. A Framework for Tangible User Interfaces. Proceedings of the Workshop on Physical Interfaces, at the 5th International Symposium on Human Computer Interaction with Mobile Devices and Services (Mobile HCI, 2003, Udine, Italy.

[5] Mazalek, A. Tangible Toolkits: Integrating Application Development across Diverse Multi-User and Tangible Interaction Platforms. Proceedings of the Let's Get Physical Workshop, at the 2nd International Conference on Design Computing and Cognition, 2006.

[6] Rogers, Y., and Muller, H.L. A Framework for Designing Sensor-Based Interactions to Promote Exploration and Reflection in Play. International Journal of Man-Machine Studies 64(1), pp. 1-14, 2006.

[7] Thimbleby, H.W. User Interface Design with Matrix Algebra. ACM Transactions on Computer Human Interaction 11(2), pp. 181-236, 2004.

[8] Raman, T.V. User Interface Principles for Multimodal Interaction. in MMI Workshop, CHI 2003.

[9] 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, pp. 223-234, 2005.

[10] S.W. Loke, Service-Oriented Device Ecology Workflows. Proceedings of the International Conference on Service-Oriented Computing, pp. 559-574, 2003.

[11] S. W. Loke, Representing and Reasoning with Situations for Context-Aware Pervasive Computing: a Logic Programming Perspective, The Knowledge Engineering Review 19(3), pp.213-233, 2004.