Middleware framework for flexible integration of new sensor types

Glen Pink, David Carmichael, Judy Kay, and Bob Kummerfeld
The University of Sydney, Sydney, Australia
{gpin7031, dcarmich, judy, bob}@it.usyd.edu.au

Abstract

One of the challenges of pervasive computing is to model location from arbitrary, available sensors. This paper explains how we have created a framework, based on the Personis user modelling framework to provide a facility for the easy addition of new sensors.

We illustrate the approach in terms of new sensors that we have added to a location modelling system: Nike+, PIR and a reed switch. Integrating a new type of sensor requires modest code. This work is important for creating future, flexible and powerful systems that can model people’s location.

1. Introduction

Location modelling is an important foundation for pervasive computing. It is the key to supporting many aspects of awareness within communities. This means that it is valuable if frameworks for modelling location be able to readily make use of a broad range of available sensor types and to be easily augmented as new sensor types become available.

This paper describes the use of the powerful rule language and accretion/resolution approach of the Personis modelling system [1] to support the simple integration of new sensor types.

Section 2 provides the background for our approach. Section 3 illustrates the process involved in adding a new sensor. Section 4 describes the rules that drive the system reasoning. Section 5 draws conclusions.

2. Background

Our pervasive computing system, Personis, models people, places, devices and sensors with a consistent data structure. Each model consists of a hierarchical name space of contexts, each containing a set of components. In the case of users the structure may be complex with many attributes of the user modelled by components grouped into suitable contexts. In the case of a simple sensor such as an infra-red detector, the structure might be very simple with only a small number of components in a single, top-level context.

The components of each model are not simple values. Instead they contain a list of evidence that has been received from other parts of the system or the environment using a "tell" operation. When an evidence item is received it is simply added to the list, or accreted, with any further processing. When an application wants the value of a particular component it uses the "ask" operation. This operation specifies a context and component but also nominates a resolver function that is to be used to interpret the evidence and produce a final value for the operation. Resolver functions may simply return the most recent piece of evidence or they might use complex reasoning over all the evidence. This approach defers the calculation of a value for a component to the "ask" stage and provides great flexibility, for example the resolver may adjust the granularity of a location value depending on who is asking.

In addition to the context/component data structure and the accretion/resolution approach to handling model values, our system also allows rules to be attached to components that are examined whenever a new piece of evidence is accreted. The rules consist of a pattern match and an action. The pattern match is typically on resolved values and if there is a match the rule initiates either another tell operation to add evidence to a model component, or a notify operation which is used to notify external applications that some event has occurred.

Sophisticated processing of sensor data can be carried using a combination of the rules and resolver functions.

The broad notion of using rules to update a location model is very natural and, unsurprisingly, it appears in many location modelling frameworks.

Indeed, even some of the earliest context frameworks made use of rules to respond to changes in system state. For example, Schilit et. al. [2] used rules to trigger events, based on context information. In that work, the action could initiate execution of an arbitrary script, clearly a powerful approach, but outside the actual framework structure.
Similarly, Activity Zones [3] used rules to map a person's location into a geographic zone.

There are many other approaches: for example, the notion of input and output widgets as a basis for building applications that drive the reasoning [4] or agents as the controlling applications [5].

There are several reviews of the range of approaches including, for example, [6].

Our approach differs because it provides a consistent structure that places simple, powerful and compact rules on any components in the Personis models and as we now illustrate, this makes it feasible to add new sensor types, linking them to very flexible reasoning that exploits arbitrary combinations of them to reason about people's location and actions such as entering or leaving a space.

For long term deployment of location modelling systems, it is important that new sensor types be easily integrated into an existing system. We now illustrate how our recent work has made this possible.

3. Processes for addition of sensors

The process for adding a new sensor to the system involves two steps: building a program to allow the sensor to send "tells" to Personis; and adding a new model for the sensor to Personis. If the sensor reports data based on the detection of transmitters, these transmitters must also be added as models to the system.

The three new sensors described below fall into two categories of reporting: the Nike+iPod sensors report identifying data, and the PIR and reed sensors report timestamps.

3.1. Nike+iPod Sports Kit

The Nike+iPod Sports kit is a set of two devices, a transmitter and a sensor, which are intended to be used so that the transmitter is in the shoe of a user and the sensor is plugged into an iPod Nano. It is possible to collect Nike+iPod Sport Kit transmission data via a serial connection to the receiver as detailed in [7]. The transmitters have an effective range of 10-20 metres, but cannot penetrate some walls. This...
range allows us to cover an area, but suited to fine-grained location detection.

We have distributed Sports Kit transmitters that are carried in a person's pocket or shoe - this transmitter fires short-range signals (a proprietary 802.11 protocol) on impact. This signal is picked up by nearby sensors which are constantly reading for data.

The Nike system provide fast location updates, as they can immediately update the system once a transmitter has come within range.

A program was written for the sensors, extracting transmitted IDs from the serial input, and sends these IDs as “tells” to the 'seen' component of the sensor model. The sensor model was built containing appropriate contexts and components, including components storing the physical location of the device and the area for which it detects.

The transmitter models are built in a similar fashion, determining what contexts and components are required and the adding appropriate rules.

### 3.2. Passive Infra-Red

Infrared sensors are frequently used in security systems to determine the presence of people.

The program to run the Passive Infra-Red (PIR) sensor is very simple and is only required to send a tell to the 'detected' component of its model indicating that a presence has been detected. The PIR model can also be simple, which (aside from default contexts/components) only requires the aforementioned 'detected' component.

Reed switches are useful for determining whether a door is currently open or closed. Their implementation is the same as that for PIRs.

### 4. Augmenting middleware to exploit new sensors

With the use of both a PIR sensor and a reed switch, it is possible to tell with considerable accuracy if a person is entering or exiting a door. By placing the PIR on one side of the door, we can determine if the PIR has detected a presence immediately before or after the door has been opened (reported by the reed switch), and hence infer that someone has entered or exited using the door. By combining this inference with identifying location data (such as the use of Nike+ sensor evidence), we can determine that a person was near the door at the time it was opened, and hence infer with some degree of accuracy that they were the one who passed through it.

Implementing this series of checks is straightforward using the Personis system, via an implementation of rules and resolvers.

The setup is as follows: a Nike+ sensor, a reed sensor and a PIR monitor a door, with the PIR monitoring only the inside of the door. A person carries a Nike+ transmitter and moves through the door.

In our scenario illustrated in figure 1 there are 5 models: the 'person' who is detected by various sensors; the Nike+ sensor 'nike_s'; the Nike+ transmitter 'nike_t' which is carried by 'person'; 'pir', the passive infra-red sensor; and 'reed_sensor', the reed sensor on the door. The scenario is as follow, illustrated in figure 1: person carrying a Nike+ transmitter approaches the door area, and the Nike sensor detects the transmitter and makes a tell to Personis, reporting that nike_t has been seenby nike_s (indicated by that arc labelled 1). Rules within nike_t fire: the location is updated by the rule on seenby (2) and the location then tells the model of the person carrying it that their location can be inferred to have changed (3).

The person then approaches the door. Depending on the direction they come from, they will set off the reed sensor then the PIR or vice-versa. In either case, these sensors will tell their models that a presence has been detected (4). The rule in reed_sensor fires, and tells the evidence tells the known nearby nike_s (5). The rule in pir fires, and makes the same tell (6).

The rule on the 'seen' component in nike_s fires every time a new piece of evidence is added to it (from either (1), (5) or (6). This rule is different from the others here as it calls a resolver (7). This resolver looks at the evidence in the 'seen' component and determines whether the evidence suggests that an entrance or exit event has occurred. If a (1) tell has been received, but a (5) and (6) have not recently (within a few seconds) then it is not considered to be an entrance/exit event. If a (5) tell has been received, and a (1) is present, then it is inferred that someone is moving through the door. If a (6) has been received just prior to this, then we can in infer that a person is exiting – if it hasn't we can infer that they are entering (and are about to set off the PIR). This is then sent as a tell to then direction component of person.

### 4.1. Rule implementation

The rules themselves are in the form “value ~ pattern: action” where the value is an ask operation on the model, the pattern is a regular expression and the action in this case is a tell. For example, the rule labelled as arc (3) above is:

```plaintext
<./location> ~ ".*": TELL <./carriedby>/location, inferred:<./location>
```
Angle brackets indicate that and ask is to be made to the enclosed component. The above reads as “when evidence is added to the location component of this model that matches the regular expression “.*”, TELL the carriedby model's location component the inference that that model is at the location that is model is located.” Rules can send tells to themselves and to other models when they receive new evidence, and send tells either as observations or inferences.

Asks can be nested, as in the example below, which is the rule for arc (2):

<./seenby> ~ ".*" : TELL ./location, observation:<<./seenby>/detectinglocation>

This reads as “when evidence is added to the seenby component of this model matches the regular expression “.*”, TELL this model's location component the observation: the seenby model's detecting location.”

Combining resolvers with rules however gives us much more power. Resolvers allow us to considered historical evidence, and hence we can make much greater inferences and do greater analyses than what is possible with rules which are limited to tells. The rules for arc (7) above implements a resolver, and it is:

<./seen> ~ ".*" : TELL <./carriedby>/location, inferred: <doorresolver!./seen>

The first part of this rule is the same as that in arc (3), however the difference lies in <doorresolver!./seen>. This calls the doorresolver on the evidence list for the component 'seen'. Resolvers can return arbitrary values, but in this case the resolver is acting as described above (determine whether someone is entering or exiting) and hence returns one of the value “entering”, “exiting” or “neither”, and the rule tells this value to the location component of the person carring the device.

By combining rules and resolvers this way we can utilise historical evidence to more accurately determine a person's location.

4.2. Code required

The above door example was implemented with only 6 short rules and approximately 10 lines of a resolver (written in python). This example demonstrates the ease of addition of new sensors with the system, and the integration of these components with each other and the system as a whole.

5. Conclusions

Possible applications for this entrance/exit data include a personalised public display which exists inside an entrance - the display can be customised to display data appropriate to the last person who entered. A “notify” (a message sent to an external server) [1] would be added on the direction field, to be fired when an entrance is detected. The ID of the person can be sent to this external system, which can retrieve the model of a person. Given this model, the display can then be modified to suit the person.

Using the available Personis framework, various sensors have been added to the system, and these sensors have been exploited to infer information greater than what the sensors individually provide. This modular approach, demonstrates the ease of integration of these sensors with the existing system, and how little code is needed for this integration. The addition of rules and resolvers allows use to further integrate the new sensors into the system and allow for a richer body of location evidence to be collected.

References