Multi-Event Handlers for Sensor-Driven Ubiquitous Computing Applications

Sean Reilly
Distributed Systems Group
Department of Computer Science
Trinity College Dublin, Ireland

Abstract—We propose an extension to the event-based programming model for sensor-driven ubiquitous computing applications. The extension consists of three abstractions: Multi-Event Handlers, Execution Policies and Event Streams. We believe that these additional abstractions simplify the task of writing applications in this domain. We present the abstractions and describe an example application from the sports domain implemented using our prototype middleware.

INTRODUCTION

Event-based programming is a popular paradigm for design and implementation of many types of computer applications. A large number of extensions have been proposed to event-based programming to make it more suitable for different domains [1], [2], [3], [4], [5], [6], [7], [8]. One notable example is the addition of filters, pre and post constraints, zones, composite events and other abstractions which have been proposed to make the event model more suitable for large scale distributed systems [2], [6], [7], [8]. The core contribution of this work is an extension of the event based programming model called the AESOP model, which is primarily designed for the domain of sensor-driven ubiquitous computing applications. Such applications typically have one or more sensors each of which provides a stream of data which must be processed continuously. In particular, applications which perform sensor fusion or context generation typically exhibit these characteristics.

THE AESOP MODEL

The AESOP (Architecture for fusing Events using Streams and executIoN Policies) model consists of events combined with multi-event handlers, event streams and execution policies. The multi-event handlers are functions that get executed in response to the arrival of one or more events. The execution policies are complex expressions that are used to decide what combination of incoming events trigger the execution of the multi-event handler. The event streams are length bounded streams of data used to supply the multi-event handler with parameters and to aid reasoning about historical sensor data.

In traditional event based systems one event caused zero or more pieces of code to be executed, typically in the form of one or more callback functions. In the AESOP model we extend this concept to allow a combination of one or more events to control the execution of the handler. The abstraction of execution policies provides the programmer with the means to specify exactly what combination of events should trigger the execution of the multi-event handler. The AESOP model is a generalisation of the standard event-based programming model, which we think is particularly useful for sensor-driven ubiquitous computing applications.

Event Streams

Event Streams are the abstraction which facilitates all communication between multi-event handlers in the programming model. They serve as input to the multi-event handler and also as the output stream. Associated with each input stream to a multi-event handler is a sliding window.

Figure 1 shows an example of some sliding windows on event streams. In the lower half of the diagram the first and third event streams have events with float data, while the second has events with integer data. The lengths of their sliding windows which are specified when initialising the event streams are 2, 3 and 1 respectively. As new events arrive they join the head of the queue pushing the last element out of the sliding windows range.

Multi-Event Handlers

Multi-event handlers can be considered a generalisation of callbacks for traditional events. Figure 1 shows a traditional event handler which executes when one event arrives and a multiple-event handler in the same diagram. Multi-event handlers generalise the behaviour of the traditional event and execute in response to multiple events. In fact a traditional event is analogous to a multi-event handler with a single
Using this middleware we have developed an application in which we call C-AESOP. Fused, creating higher level sensor streams such as S6, S7 and S8. The sensor data is continuously being refined and demonstrated in figure 2. Three layers of multi-event handlers execute given new event stream data. We can decide to execute the multi-event handler using this new event or wait until all streams have new events, or some combination of the two options. Execution policies are the abstraction used to represent these decisions in the programming model.

**Composition of Multi-Event Handlers**

The ability to compose multi-event handlers into chains is a very useful and elegant feature of the AESOP model. The outputs of multi-event handlers are themselves event streams and can become the input to multi-event handlers. This is demonstrated in figure 2. Three layers of multi-event handlers are shown. The sensor data is continuously being refined and fused, creating higher level sensor streams such as S6, S7 and S8.

**EXAMPLE APPLICATION IN SPORTS DOMAIN**

In order to evaluate the AESOP model we have implemented a prototype middleware in C which we call C-AESOP. Using this middleware we have developed an application in the sports domain for the sport of squash. Squash is a sport played indoors in a small static area between two athletes using rackets and a ball[9]. We have augmented a squash racket with several sensors in order to provide advice to the athlete on how to improve their swing. The angle of contact between the face of the squash racket and the ball is of interest to advanced squash players. We have augmented the squash racket with an orientation sensor and a sensor which detects ball contact. Figure 3 shows the events and multi-event handlers used in the application. When a ball contact event occurs the orientation of the racket is calculated and the resulting orientation event is produced. We are in the process of adding load sensors to the squash players shoes. This would allow us to analyse the athletes weight distribution as they perform the swing.

**FUTURE WORK**

Sliding windows are the only history mechanism in the present model. Various other mechanisms such as tumbling windows[10] may be included in the future in a similar manner to history mechanisms which have proved useful in other domains [11]. At present the storage policy for event streams is to store all data on the stream. Other data storage policies may also be required.

**REFERENCES**


**Figure 2. Composition of Multi-Event Handlers**

**Figure 3. Squash Application**