

Mobile Bristol Application Framework

Richard Hull
13 December 2002

1 Introduction

Mobile Bristol is a collaborative project to establish an experimental, real-world testbed for research in pervasive computing. The project consists of three broad, interlinked strands:

- The provision of infrastructural elements such as a wired and wireless network, clients, servers, and instrumentation.
- The experimental development and deployment of mobile, context-sensitive applications and services.
- The exploration of user responses to the experiences and values provided by those applications.

This document concentrates on the second of these strands to outline a proposed framework for application development in Mobile Bristol. It will inevitably touch upon other aspects of the project but will not attempt to provide any comprehensive overview of their activities. The document is structured as follows: In the next section, we will review the overall aims and philosophy of the framework, then we will focus in turn on clients, servers, behavioural specifications, and authoring tools. Finally we will discuss issues arising from or not covered in the remainder of the document.

2 Overview

The application framework described here is not, of course, the only way in which to develop applications for Mobile Bristol. Indeed, we expect programmers to explore a range both of application types and of ways of making those applications. However, we do hope to provide a rapid and convenient development path for researchers, creatives, schoolchildren and others who are interested in exploring new values for context-sensitive applications and services, but may not necessarily have programming skills. In this light, we can identify the following objectives for the framework:

- It should support a diverse set of applications, many of which will not have been identified before the framework appears.
- It should not require developers to commit to a particular application architecture.
- It should allow new applications to be introduced without any need to re-program client devices already released into the Mobile Bristol environment.
- It should allow rapid development of new applications by providing a set of in-built capabilities.
- It should allow (some) applications to be developed without programming expertise.

Given these objectives, the overall design philosophy adopted in the framework is to separate the specification of application behaviour from the implementation of the client devices. In particular, applications in this framework are represented by explicit specifications of

context-sensitive behaviour that are downloaded and interpreted by client devices as appropriate, as shown in figure 1.

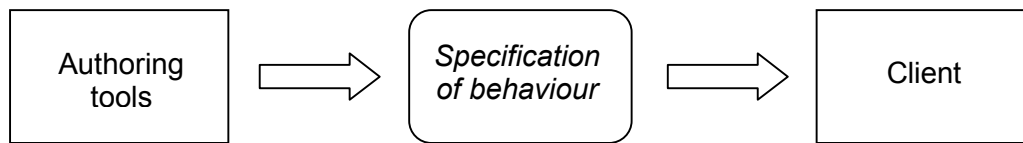


Figure 1 Developing applications in the proposed framework

This approach should be familiar from the World Wide Web, where explicit specifications of web pages (or Flash movies or other web content) are downloaded and interpreted by general purpose web browsers. It is also the approach taken in the first generation of Mobile Bristol clients developed by Hewlett-Packard Laboratories in which downloadable specifications were used to define situated soundscape applications.

In the proposed framework, the specifications representing the behaviour of an application will be defined in a mark-up language derived from XML, and will largely consist of descriptions of how to handle events that can arise in the client device's environment. We will return to this topic in section 4. For now, it is sufficient to say that the specification language can be used to implement a variety of standalone, client-server and peer-to-peer application architectures, and that though biased towards event-driven content delivery, the language allows arbitrary application logic through scripting extensions. Thus, we claim that this approach should allow the framework to satisfy at least the first three of the objectives stated above. Satisfying the other two depends on the details of the client and language capabilities, and our skill in developing appropriate authoring tools.

3 Clients

We can imagine a variety of client devices in Mobile Bristol, though it might eventually be useful to distinguish between two broad classes:

- Mobile clients, such as cyberjackets, PDAs or phones, that are carried (or worn) by the user as they move around the environment
- Fixed clients, such as displays, that are situated in and augment the environment.

For the moment, we will ignore this distinction while trying to remember that it might have significant consequences later in terms of, say, network connectivity, power budgets, and personalization.

3.1 Hardware

Whether mobile or fixed, we assume that a client device potentially has the set of components shown in figure 2. Of course, this is a gross simplification of the actual architecture of any particular client, but it serves to illustrate the main capabilities of client devices. In particular, it implies that clients can potentially sense their environments and receive, store, process, render, capture and transmit a variety of multi-media content. However, it is important to note the word "potentially" in the sentence above. We do *not* assume that all clients will have

all of these capabilities. For example, we can easily imagine audio-focused devices that do not have a screen, or a device with a large cache instead of a permanent network connection.

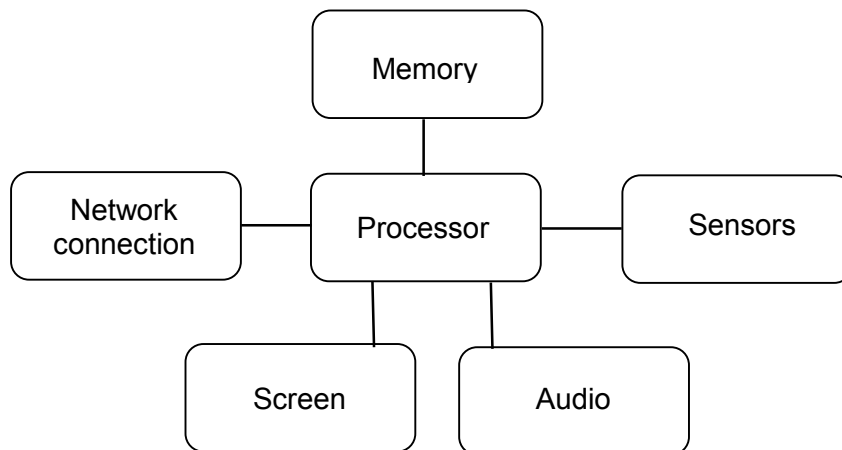


Figure 2 Client hardware components

One consequence of this non-assumption is that client profiling may be necessary to ensure that only specifications that make sense for a particular client configuration are downloaded. In this case, we would expect to adopt (or adapt) one of the emerging profiling standards such as CC/PP.

3.2 Software

Naturally, any software implementation that enables a client device to correctly operate according to a downloaded application specification is acceptable in Mobile Bristol. However, the proposed framework will also include an exemplar implementation whose architecture is shown in figure 3. In addition to the underlying requirement to download and interpret behavioural specifications as described above, this architecture is primarily influenced by the adoption of an *event-driven programming* approach, in which application behaviour is modelled as a set of responses to events arising within the client's environment. This approach, familiar from systems with asynchronous interfaces, provides a simple model for non-programmer developers, and allows the client to integrate user actions, changes in environmental context, and incoming notifications from other systems in a uniform manner.

The proposed client software includes the following principal modules:

- **Event manager**
As suggested by our previous comments, the event manager is at the heart of the client architecture. It is responsible for receiving events, interpreting them according to the currently loaded application specification, and orchestrating action.
- **Context manager**
Responsible for maintaining an up-to-date and relevant set of behavioural specifications on the client.
- **Data cache**
Responsible for downloading, streaming and caching remote data.

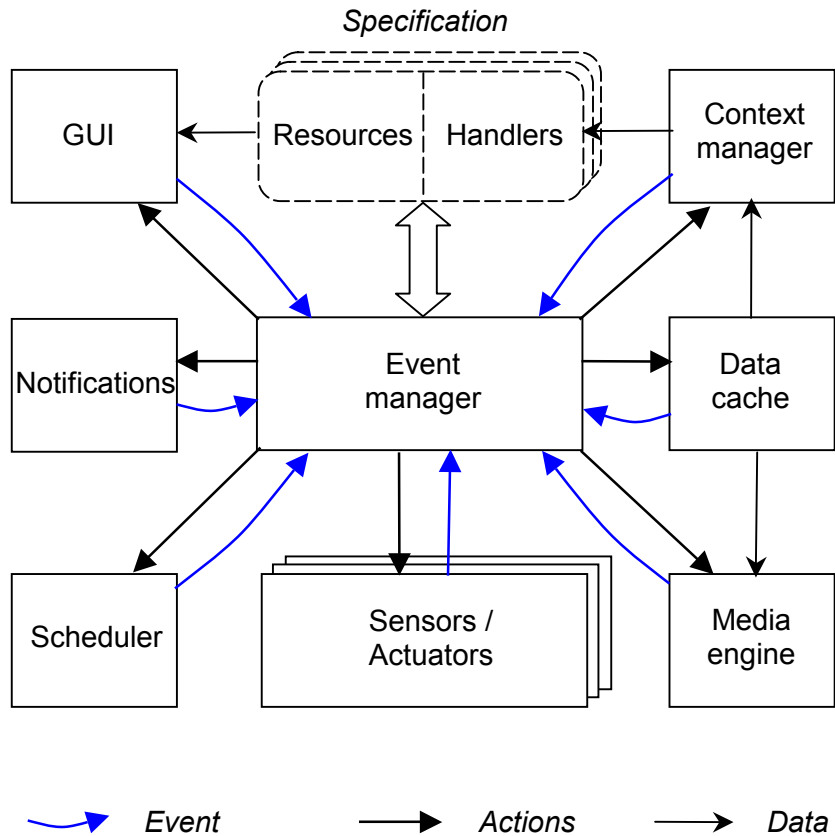


Figure 3 Client software architecture

- **Media engine**
Responsible for rendering and capturing multi-media objects.
- **Sensors & actuators**
Responsible for sensing and acting upon the client's physical environment.
- **Scheduler**
Responsible for maintaining a schedule of impending actions and raising appropriate alarms.
- **Notification engine**
Responsible for sharing state and events with other entities, including other clients. In the first instance, notifications will be distributed via a publish & subscribe scheme.
- **GUI**
Responsible for direct user interaction with the client.

A short example might help to give a feel for the flows of events, actions and data between these modules: Imagine a collaborative, interactive game defined in a particular specification downloaded by the context manager on the basis of the client's current location and hardware profile. After loading the specification, the context manager issues an appropriate event to the event manager which responds by finding and interpreting the corresponding *onLoad* handler in the specification. This, in turn, might instruct the event manager to invoke the data cache to download and store a handful of short audio files for later use. Now the game starts. As the client's user moves around the game space, the event manager starts to receive a series of

location events from, say, an ultrasonic location sensor. The event manager interprets each location event by searching for a matching *onLocation* handler in the current context and invoking the actions specified therein. In this case, the resulting behaviour might be to instruct the media engine to play one of the pieces of audio cached earlier. Shortly after, an external notification might arrive from the client device belonging to another player, again causing an appropriate event to be posted to the event manager. Once more the specification is used to provide a handler, perhaps this time to activate an indicator LED on an off-board peripheral. At that point, the user might hit an interface button, another event is generated, the event manager again swings into action, and so on the system goes.

3.2.1 Platform

It is intended that the exemplar client software will be implemented for a variety of platforms corresponding to the hardware devices of interest. In practice, this means supporting the Windows CE, Linux and Symbian operating systems. The first implementation of the software is likely to be in C++ to leverage existing code investments but we would also like to have a Java implementation fairly soon. The C++ implementation will use a hardware abstraction layer to facilitate porting across the different platforms.

Eventually, the intention is for the client (and any other) code to be open sourced. In the first instance, access will probably be mediated by a CVS server restricted to Mobile Bristol participants.

4 Servers

The objective to allow diversity in application architecture implies that we should have as few requirements as possible for mandatory servers in Mobile Bristol. However, the project will provide a few server types whose existence can be assumed by application developers using the framework:

- Network infrastructure servers such as DHCP and DNS.
- A directory server that maps application specifications to contextual information such as location. This acts as a kind of search engine for clients trying to discover which applications are appropriate to their current situation and capabilities. Note the specifications themselves will be stored on standard web servers.
- A publish & subscribe event notification server for distributing messages between clients, and between clients and servers.
- Standard web servers serving application specifications and standard web content.
- (Possibly) media servers supporting streamed media.

Note that the very simplest applications might not use any of these servers. Others might just use DHCP, DNS and the directory server. Yet others might introduce a range of supplementary hotspot, proxy, game, or other servers, some of which might migrate over time into the base provision.

5 Behavioural Specifications

As stated earlier, the behaviour of clients in (this form of) Mobile Bristol is determined by an explicit specification that is downloaded from some server on the Internet. In this section, we introduce the main language concepts used in those specifications, as summarized in figure 3.

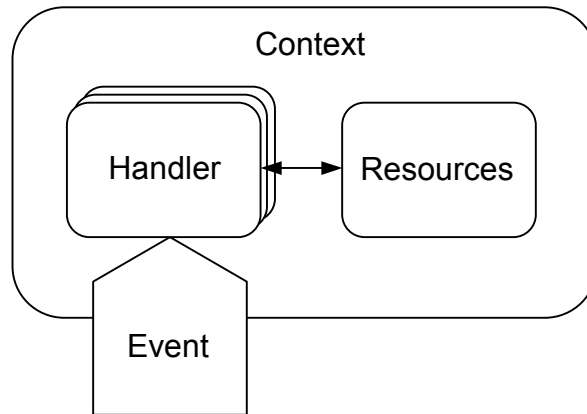


Figure 3 Relationship of main language concepts

The proposed specification language is described more detail in other documents. For now, the main concepts of which to be aware are:

- **Contexts**
A *context* is a specification of behaviour written in the context language. It consists of a set of *event handlers* and *resources* that collectively define the response of a client to its user and environment. Contexts may be nested to allow, for example, contexts specifying different levels of a game to share a common set of background behaviours. Contexts are accessed by a URL.
- **Events**
An *event* represents some discrete occurrence in the user interface or environment such as a button being pressed or the detection of movement of the user in some physical space. Events can be generated on the client by a variety of sources including sensors, interface widgets, timers and incoming notifications from other entities. The specification language will contain both a set of built-in events and the ability to define new events.
- **Handlers**
A *handler* is a response to an event and consists of a set of *actions* guarded by *conditions*. Actions consist of calls to built-in functions or to extension functions defined in a scripting language. They may be executed in an arbitrary order or in a defined sequence. Conditions are represented by Boolean expressions containing standard operators, functions, and context-defined variables.
- **Resources**
A *resource* is any of a number of objects specific to that context that may be used within event handlers. Examples include *variables*, *regions*, *scripts*, *notifications*, *virtual events*, and *media objects*. We assume that most of these are self-explanatory. Regions are ranges that relate to specific sensed data, such as physical areas, or time periods. Scripts are code

fragments in some scripting language, such as JavaScript, that potentially extend the specification language towards a full-blown programming language.

An application developer is thus primarily responsible for authoring one or more contexts that include handlers for the client events of interest. This may require the developer to define various resources for use in those handlers and refer to other Web resources and to other contexts. The completed contexts are then published to a server from where they may be downloaded by clients over the Internet.

Figure 4 shows an example of a simple soundscape context. Note that the particular language syntax used in the example is provisional and that the application logic is largely illustrative.

```
<Context>
  <Resource>
    <Region name="january" x="100" y="200" range="100" />
    <Region name="february" x="500" y="200" range="100" />
    <Variable name="heardJanuary" type="bool" value="false" />
    <Notification type="success" >
      <Param name="deviceId" type="integer" />
    </Notification>
  </Resource>
  <Handler>
    <OnEvent type="MB_REGION_ENTERED" params="regionID">
      <Action condition="regionID==January && heardJanuary==false">
        <Play type="audio" url="/server/jan.mp3" loop="true" />
        <Assign name=" heardJanuary" value="true" />
      </Action>
      <Action condition="regionID==february">
        <Play type="audio" url="/server/feb.mp3" loop="true" />
        <Send type="success">
          <Param name="deviceId" value="MB_MYID()" />
        </Send>
      </Action>
    </OnEvent>
  </Handler>
</Context>
```

Figure 4 A simple example of a context

This example breaks the context into two broad sections: resources and handlers. In the resources section, we declare two positional regions, a state variable, and a notification type. In the handler section, we declare a single handler for the MB_REGION_ENTERED event which is raised when the client moves into one of the declared regions. The handler consists of two blocks of actions, each of which is conditioned by some expression. In this example the conditions happen to be mutually exclusive but that need not be the case. The actions associated with entering the January region are to start playing an audio stream and to note a change of state. The actions associated with entering the February region are to start playing a different audio stream and to send an appropriate notification to whoever might be listening. The notification carries the client's id which is retrieved by the built-in function (or script) MB_MYID().

The example is not, of course, exhaustive. It does not show, for example, how scripts are defined, or that actions can be sequenced, or how contexts can be combined. However, we hope that it is rich enough to give a feel for how the specification language might be used.

6 Authoring Tools

One of the design goals for the specification language is to define a limited set of capabilities that combine into a simple conceptual model for non-programmer developers. However, even if this aim is achieved, writing verifiable XML will remain a daunting task. Consequently, the proposed application framework will include the suite of authoring tools illustrated in figure 5.

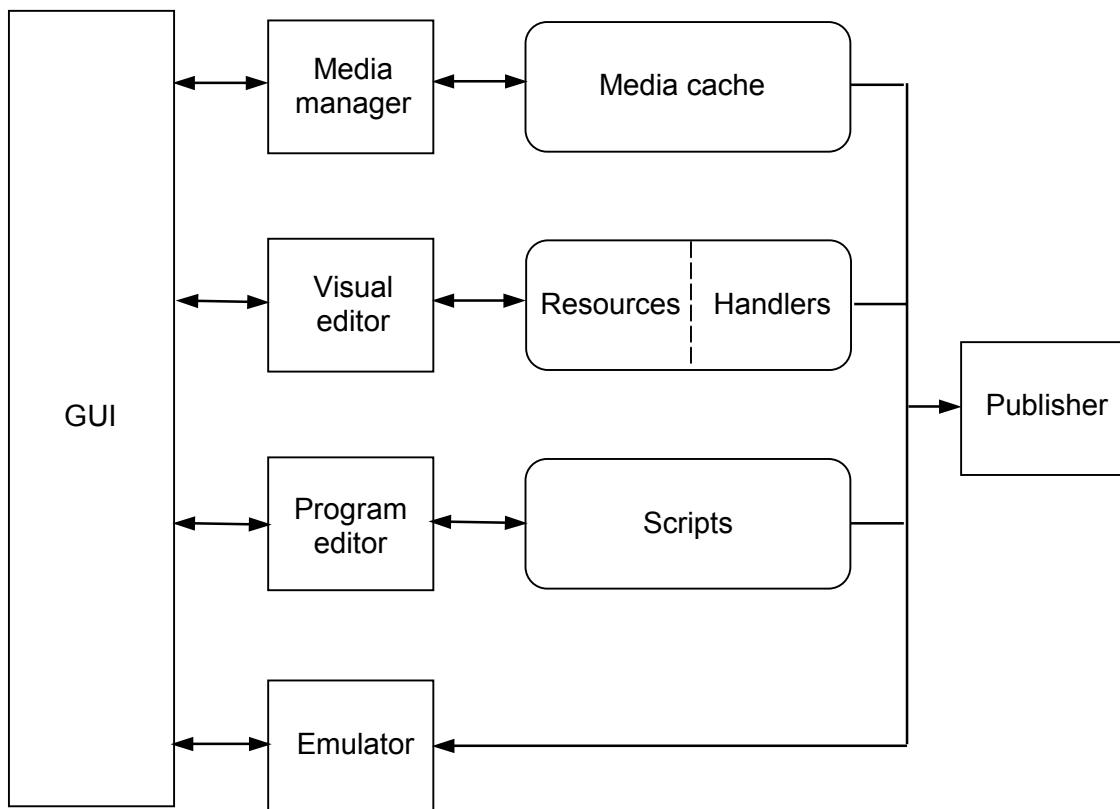


Figure 5 Authoring tools

The central tool for most authors (or developers) will be a visual editor allowing contexts to be created through a point & click graphical interface. This may have multiple views enabling, for example, physical regions to be laid out on a map of an area, and the use of dialog boxes to declare resources or define event handlers. If the editor is successful, the author should never need to see the underlying XML being produced.

For many applications, such as simple soundscapes, this visual editor will be sufficient. However, other applications may require features that are beyond the (intentionally) limited specification language. The program editor is provided to allow developers to extend the specification through a scripting language, for example by defining new action functions. The

relationship between the two editors is intended to be similar to that in, say, Macromedia Director, in which developers can produce a range of movies through a graphical editor but drop down into editing Lingo directly if the need arises. We assume that both editors will be constrained (as far as possible) to generate syntactically correct XML.

As the proposed framework is tuned for media intensive applications, we also envisage a media management tool that allows authors to collect and organize a collection of media objects (or links to those objects). Using these tools, application developers eventually end up with one or more contexts and a set of media objects that are intended to represent the desired application behaviour. Some testing can then be done via an emulator that will interpret events triggered by the developer, for example by moving a representation of a user around the map of the area in question, and perform the appropriate actions. As well as reinforcing the developer's confidence in the logical correctness of the specification, the emulator helps provide a feel for the experience delivered by the application. Of course, the real test must wait until the application is deployed and we envisage a publisher tool that distributes the specification and media to chosen servers, and updates the directory server accordingly.

7 Discussion

The proposed framework is described in this document at a deliberately high level in order to convey the overall design philosophy and structure. Subsequent documents will introduce much more detail on the specification language, exemplar client, and authoring tools. However, even this outline raises several issues that would be fruitful topics for feedback and discussion.

One such issue concerns the coverage of the framework, in the sense of the range of applications that can be built and the variety of client devices that might be covered. For example, the framework currently provides no support for synchronous voice communication. That is not to say that applications built within the framework cannot provide this capability, but rather that their developers will not get any particular help from the framework in doing so. The existence of a scripting language, and the ability to introduce new, specialized clients means that any capability can potentially be included in an application. However, the presence of existing clients in the environment and the scope of the specification language and authoring tools will tend to dampen such innovation in those using the framework. Consequently, it is important to ensure that the framework includes the features that we feel will be important over the next couple of years. What, if anything, should be added to the coverage implied in this document?

Related to this issue is the question of who should use the framework. As stated before, the framework is supposed to provide assistance, not force conformance. In other words, any Mobile Bristol developer is free to use the framework or not. However, we want to encourage its use for a couple of reasons. First, the idea behind Mobile Bristol is to provide an experimental vehicle that facilitates research in pervasive computing by allowing a new idea to be quickly implemented and tested in the context of an existing infrastructure. For example, a project student with a new idea for, say, sensing location, should be able to deploy and investigate a prototype sensor without having to build all the rest of the stuff that would be needed to provide a realistic testing ground. The application framework is part of that infrastructure. Secondly, the framework potentially provides an integration point for contributions from the various participants in Mobile Bristol. As new capabilities are

researched and developed, their introduction into the framework means that they become available to others working in this context. With luck, synergies emerge and Mobile Bristol begins to become more than the sum of its parts.

A related issue is how the framework will be distributed, extended and maintained. The intention here is to adopt the open source model. The specification language, example code, and authoring tools will be put into the public domain for community ownership and further development. One question is whether this community should be truly open or restricted for some time to Mobile Bristol participants. A second is whether certain aspects of the framework, particularly the specification language, should be subject to some standardization procedure such as that administered by W3C. Perhaps this will be a matter of maturity, in which truly open status and standardization will be sought after a period in which the framework is shaken down by the Mobile Bristol community.