

EQUIP: a Software Platform for Distributed Interactive Systems

Chris Greenhalgh

The Mixed Reality Laboratory,

The University of Nottingham

Nottingham NG8 1BB, UK

Tel: +44 115 951 4221

cmg@cs.nott.ac.uk

ABSTRACT

EQUIP is a new software platform designed and engineered to support the development and deployment of distributed interactive systems, such as mixed reality user interfaces that combine distributed input and output devices to create a coordinated experience. EQUIP emphasises: cross-language development (currently C++ and Java), modularisation, extensibility, interactive performance, and heterogeneity of devices (from handheld devices to large servers and visualisation machines) and networks (including both wired and wireless technologies). A key element of EQUIP is its shared data service, which combines ideas from tuplespaces, general event systems and collaborative virtual environments. This data service provides a uniquely balanced treatment of state and event-based communication. It also supports distributed computation – through remote class loading – as well as passive data distribution. EQUIP has already been used in several projects within the EQUATOR Interdisciplinary Research Collaboration (IRC) in the UK, and is freely available in source form (currently known to work on Windows, IRIX and MacOS-X platforms).

INTRODUCTION

The development of novel interactive devices and the deployment of mobile communication infrastructures have fuelled a growing focus on ubiquitous interactive systems that support people within real world environments. These systems place digital information in physical spaces [28] focusing on the delivery of information to users through a heterogeneous collection of devices ranging from handheld and wearable computers to large embedded displays. The majority of these systems have exploited a sense of location as a contextual cue to drive the interaction.

An equally significant trend has been the growth in the number and diversity of collaborative virtual

environments to manage cooperative interaction [2, 12, 26]. Just as ubiquitous computing environments exploit real world location, these systems exploit a sense of location within a virtual world as a contextual cue for interaction. However, despite significant similarities, these two research approaches have often tended to be seen in opposition to each other, with ubiquitous computing embedding computers with the world of users, and virtual environments embedding users within a computer generated world [16].

As part of our ongoing research we are exploring the advantages to be gained through the convergence of these approaches, allowing a collaborative virtual environment to be overlaid on top of a shared physical space. A number of key advantages motivate our desire to combine the physical and virtual to support interactive systems:

- The ability to exploit the coextensive virtual world as a ‘behind the scenes’ resource for coordinating and managing devices and interaction in the physical space.
- The opportunity to develop applications that span the physical and digital realms, for example that require collaboration between field operatives and control-room personnel.
- The chance to support new kinds of interactive experience, combining elements from virtual worlds (e.g. rich media content, high interactivity) with varied modes of access over extended geographical areas and periods of time (e.g. across a city, over a period of days or weeks).

Our ultimate goal is to develop a rich interactive experience that combines physical and digital space, with digital interaction becoming increasingly interwoven with everyday interaction in the physical world. This paper presents the EQUIP platform [9], developed to support the merging of physical and virtual environments as part of the EQUATOR Interdisciplinary Research Collaboration (IRC) in the UK [8].

EQUIP is freely available (including source) for other practitioners to make use of [9]. The rest of this paper gives an overview of EQUIP, and its key elements before

providing some brief examples of how EQUIP has been used to date, illustrating some of the aspects described. We conclude by relating EQUIP to other work and identifying some of the next areas for development and exploration.

MERGING THE PHYSICAL AND DIGITAL

The EQUATOR IRC is exploring the relationship between the physical and the digital, especially in everyday life. In particular, we are working towards a common understanding of mixed realities [16], which ranges from Collaborative Virtual Environments (CVEs) at one extreme, to tangible, embedded and wearable systems at the other extreme.

This has involved the construction of a range of different environments ranging from City and museum guides [15] to playful environments for school children [19]. EQUIP provides a common platform that allows applications to be developed where interactions in the physical world may have a direct digital effect in a virtual world, and interactions in the virtual may become manifest in the physical. Physical and virtual worlds each have distinct characteristics and place different demands on a supporting infrastructure. Virtual worlds are characterised by:

- Significant amounts of structured state information (describing the virtual world and the users within it);
- Rapid updates on some data (e.g. user movement and interaction);
- Demanding performance requirements (e.g. for realtime rendering and interaction);
- Tight integration of multiple media (e.g. embedded realtime audio and video streams).

On the other hand, devices placed in the physical world are characterised by:

- Extreme diversity of devices, including diverse computational, storage, input and display capabilities and variations in networking (e.g. bandwidth, reliability, disconnection);
- Coordinated use of collections of devices, both by the same person (at the same or different times) and by different people in the same physical environment.

As well as addressing these demands, EQUIP must also facilitate interfacing to a range of existing and independent software-based systems and services to allow digital devices to access as wide a range of resources as possible. In architecting and implementing EQUIP we make the following assumptions:

- Applications and patterns of use will typically span and combine multiple devices, in both local and geographically separated locations.

- Various kinds of computing-capable devices will be involved, including PCs and workstations, laptops, handhelds, and other mobile and wearable devices.
- Networking will be realised by varying combinations of wired & wireless infrastructures.
- Input & output devices may be distributed over multiple computers, and various IO devices and modalities may be brought together in a single interaction.

We have also specified a number of additional requirements:

- Comprehensive support for both C/C++ (for performance, e.g. 3D graphics) and Java (for code portability and easier programming).
- Timeliness and throughput sufficient to support interactive applications, such as CVEs.
- Extensible software organisation, making it relatively easy to add new data (object) types, implementations, and applications.
- Support for modular and loosely coupled application development, encouraging and supporting reuse.

Our focus in this paper is on the technical and architectural elements of EQUIP and especially its approach to data sharing and coordination. Three elements are key to EQUIP:

- **A core runtime structure** that supports dynamic code loading.
- **A Module Structure** that supports the development-time structuring of code.
- **Module Naming Conventions** that allows direct mappings between C++ and Java.

The Core Runtime Structure

As illustrated in figure 1, EQUIP depends on a core runtime system that supports modularisation and code loading (below the solid line). For C/C++ this is Bamboo [1,29] for modularisation and code loading, running over Netscape Portable Runtime (NSPR) [17] for cross-platform portability. For Java this is a regular JVM (Version 1.1.3 or above, including Personal Java). EQUIP itself is organised as a number of modules within this framework. These include:

- Core runtime modules supporting standard facilities such as memory management (for C++), and serialisation (dual language).
- Various utility modules, e.g. for vector and matrix maths, TCP and UDP networking.
- A shared data service, which incorporates ideas from tuplespaces with our previous work on efficient and timely data sharing for CVEs (see following section).
- Common interface and type definition modules, e.g. for renderable objects, audio and video session

descriptions, and device input (analogue, digital, etc.).

- Implementation modules, including specific implementations of common interface types (e.g. renderable 3D objects) and applications (both client and server-type processes).

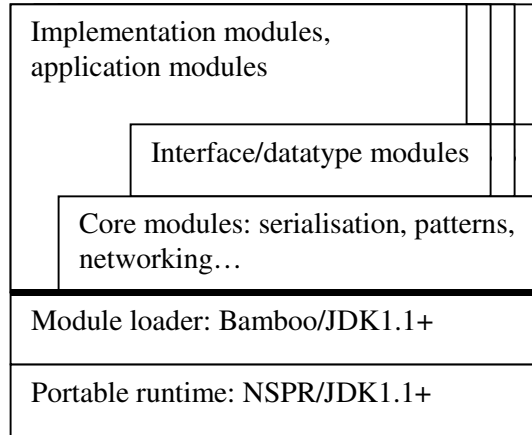


Figure 1. Runtime structure of EQUIP

The Development-Time Module Structure

Modules are a fundamental part of EQUIP, providing the basic support for software organisation, compilation and loading. Figure 2 shows the structure of a typical EQUIP module.

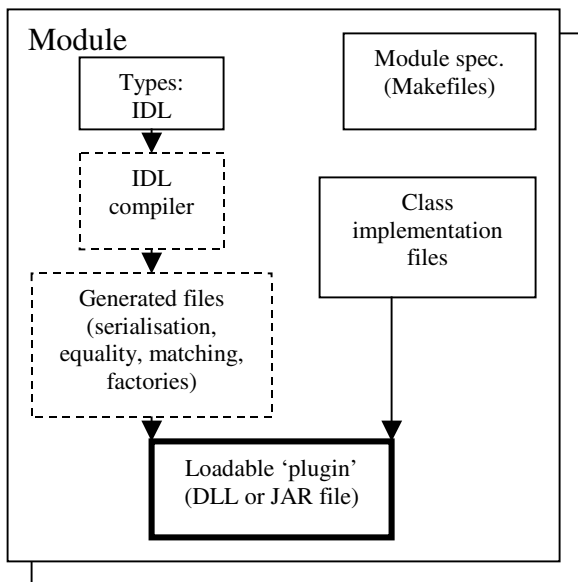


Figure 2. Development time structure of an EQUIP module

Part of the structure is a standard Bamboo module, comprising: implementation source code, make files (which also define the module name, initialisation entry points and dependencies) and compiled DLLs (for C++) and JAR files (for Java). Bamboo provides a framework that supports:

- Compilation of modules, including both C++ and Java content; and
- Dynamic loading of modules at runtime, which is currently used only for C++ EQUIP applications.

In addition, EQUIP adds: programming-language independent type/class definition files; and the class definition files auto-generated from these by the EQUIP IDL (Interface Definition Language) compiler, “eqidl”. This currently supports:

- The Object By Value (OBV, or “valuetype”) subset of CORBA IDL [18] (but not local or remote interfaces at present);
- Language bindings for C++ and Java;
- Auto-generation of serialisation, equality testing, type-safe casting (narrowing) and pattern matching.

Although the IDL used is (a subset of) standard CORBA, the generated code and language bindings differ in a number of respects: the serialisation is proprietary, and somewhat closer to Java serialisation; a pattern matching operation is generated, which is used extensively in the EQUIP data sharing service; and some details of the language binding are non-standard.

The main benefits of using an IDL to define classes are that: the ‘same’ classes can then be used in the same way in both C++ and Java; instances in one process – C++ or Java – can be serialised and read in by any other process in either language; and standard facilities can be provided by the IDL compiler without the need for additional programming, such as pattern matching and standard object factories.

Module Conventions and Code Loading

As already noted, Bamboo supports dynamic loading of modules. However, in order to support generalised class loading programmers must also observe certain naming conventions:

- All IDL declarations within an EQUIP module must be in a single CORBA module namespace (e.g. “module example { module myModule { ... }”); this will generate C++ classes in the corresponding C++ namespace (e.g. “::example::myModule”), and Java classes in the corresponding Java package (e.g. “example.myModule”).
- The name of the Bamboo module must match the IDL module name, with underscores replacing double colons (e.g. “example_myModule”).
- All module names must be unique, and so each module’s IDL declarations must be in a unique namespace.

These conventions allow the EQUIP C++ runtime system to map from class names to the corresponding Bamboo module. When the module is loaded it registers object

factories for all of the classes that it contains (using C++ static constructors), allowing the runtime system to load and instantiate arbitrary classes at runtime. At present the EQUIP Java runtime directly loads named class using Java's native class loading facilities.

EQUIP DATASPACES

A key advantage of EQUIP is the use of the data sharing service – referred to as a dataspace – to provide a bridge between collaborative virtual environments and multiple real world environments. The data space allows information elements to be simultaneously available through the virtual environment and through devices in the physical world. This arrangement is shown in figure 3

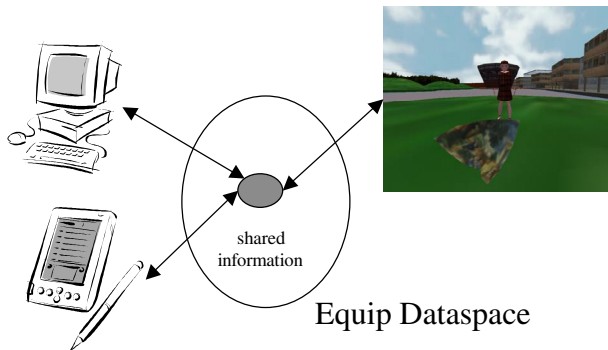


Figure 3. Sharing data between virtual and physical

Most applications of EQUIP revolve around the use of the equip data service to make information available across a community of devices and virtual environments. This shared data service has a number of objectives:

- Easy introduction of new data types.
- Tightly integrated support for both *event*-oriented and *state*-oriented data distribution.
- Support for loosely coupled data sharing, exploiting pattern matching, inspired by tuple spaces [10] and general event systems such as Elvin [20].
- Timeliness and throughput sufficient to support interactive applications, such as CVEs.
- Support for intermittent connectivity, limited bandwidth and variable loss, for example over wireless networks.

In the current implementation any application can create new dataspace servers, or dataspace clients that attempt to connect to a particular dataspace server. A simple EQUIP name service allows dataspace servers to be identified using a URL scheme of the form "equip://host:port/name".

The EQUIP data service uses a combination of event distribution and state sharing to propagate the effects of interaction with shared information.

Event Distribution

At one level, the EQUIP data sharing service can be used as an event distribution system, comparable to Elvin, for

example. There is a standard event base class, `equip.data.Event`, which the application programmer can subclass in other modules to create their own event types. An event is sent using the `addEvent` operation on a dataspace (client or server). Other clients (or the server or other portions of the originating client) can subscribe to events by creating an event pattern object, which includes one or more template event objects. Whenever an event is sent that matches the template(s) a corresponding callback is invoked. This is illustrated in figure 4, omitting the details of the subscription and event distribution mechanisms, which are described later.

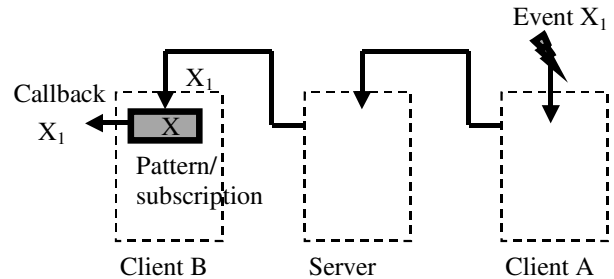


Figure 4. Simple event distribution in a dataspace.

Distribution of events from the server to each client is demand driven, i.e. events are only forwarded to clients that have subscribed to them. The current implementation forwards all events generated in a client to the server, excluding specifically local-only events. The realisation of demand-driven forwarding depends on state sharing, described next.

State Sharing

Pure event systems can be used for state sharing, however additional protocols must be overlaid on the event system or additional constraints must be enforced. For example, a late joining client may send an event asking to be informed about existing state; the other clients must respond with events describing the currently existing state. In this case, race conditions can arise in which the joining client receives updates before they have received the reply that describes the state. Alternatively a late joining client will not discover the existence of stateful elements in the environment until they generate an event, and each event must contain sufficient information to fully describe the implied state.

In the EQUIP data sharing service we have integrated state sharing into the data service itself, giving a hybrid state/event distribution service. This is intended to allow the application programmer to use events for ephemeral actions and state for stateful elements. For example, a button is better represented as an element of state that is subject to change than as a sequence of events, because the button exists and has a state (on or off) even if a new observer has not yet seen its state change.

The data service defines three core event types: `equip.data.AddEvent`, `equip.data.UpdateEvent`, and `equip.data.DeleteEvent`. These are used and generated by applications and by the dataspace itself to represent the creation, modification and removal of state, respectively. The data server also defines an abstract base class for all shared state, `equip.data.ItemData`. Application programmers can share new forms of state by creating new modules containing application-specific classes that extend this.

The extensions to the data service to support state are two-fold. First, each data service client or server incorporates a memory-local database – the dataspace – in which it maintains copies of currently known data items (i.e. instances of subclasses of `equip.data.ItemData`). When an add event is received, after calling any matching callbacks, the corresponding data item is added to the local dataspace. Similarly, update events update the item in the dataspace and delete events remove the referenced item.

The second extension is that the data service will optionally synthesise add and delete events when corresponding event subscriptions are made or removed. In particular, an “item monitor” subscription expresses interest in add, update and delete events on a specified array of data item templates. When an item monitor is added to a dataspace the dataspace will generate subscription-specific add events for all matching data items already in the local dataspace, and then an item monitor is removed from a dataspace it will generate subscription-specific delete events for all matching data items remaining in the local dataspace. This ensures that the user of an item monitor will always see exactly one add event for each matching data item (either when the item monitor is created, or when the item is subsequently added to the dataspace), zero or more update events, and finally one delete event (either when the item monitor is destroyed, or when the item is actually deleted from the dataspace). This is illustrated in figure 5

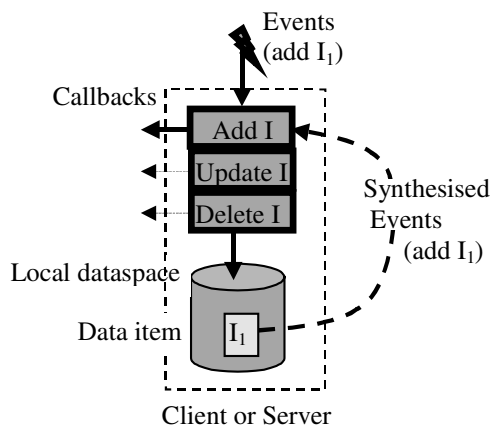


Figure 5. Integration of events and dataspace.

Distribution and Replication

The previous description has not directly addressed how events and data items are replicated between data service client(s) and the corresponding server. In the current implementation every client sends all events to the server (including synthesised add and delete events on connection/disconnection). The server passes the event copies to its own `addEvent` operation. Consequently all (non-local) data items in each client are replicated in the server’s dataspace, and all events in a client are also seen by the server.

To achieve demand-driven distribution and replication the subscriptions that are added to a data service are reified (given a concrete internal representation) as event pattern data items within that dataspace. Like all other data items these are therefore replicated to the server. The server uses this information to create equivalent local subscriptions in the server, the callbacks of which forwards the matching events to the corresponding client.

For example, as illustrated in figure 6, a client might subscribe to events of type T (1). This will cause the client data service to also create an event pattern data item including a copy of this event template (2). The add event representing this created data item will be copied to the server, which will create an equivalent subscription to events of type T on behalf of that client (3). Another client which emits an event of type T (4) will send a copy of that event to the server (4b), which will re-emit the event there. This will match the subscription held on behalf of the client, and the event will be forwarded to that client (4c). The original client will receive the event and will re-emit the event itself. This will match the client’s original subscription, calling the corresponding callback to allow client code to execute. Precisely the same mechanism – applied to add, update and delete events – causes data items to be replicated between clients and the server.

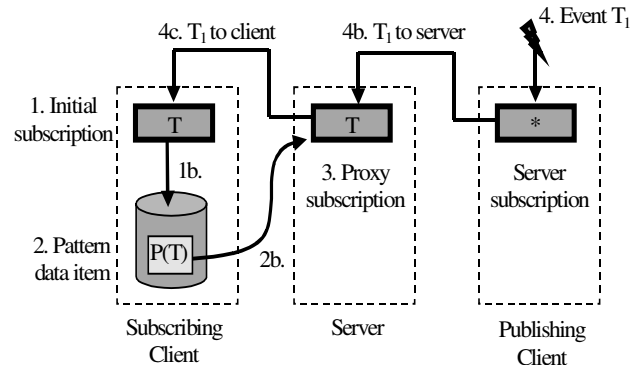


Figure 6. Demand-driven Event Distribution.

Discussion

Subscription to events and data items is based on the built-in pattern matching between all IDL-defined classes (in particular events and data items). In general pattern matching approaches to communication encourage and

support loosely coupled application development and construction: the data server can be used almost like a blackboard system, in which clients obtain input – irrespective of its origin – by pattern matching over a data service’s events, and publish output – independent of its final consumer(s) – by sending it back to a data service.

The basic event distribution style is relatively standard in its use. However the state sharing style has a combination of distinctive features.

- Like tuplespaces, items can be added to the dataspace (add, c.f. “out”), removed from the dataspace (delete, and also event patterns which delete data items on matching c.f. “in” or “collect”), or copied from the dataspace (item monitors and other event patterns comparable to “copy-collect”).
- Unlike pure tuplespaces, every item must have a globally unique identifier. Consequently every data item is unambiguously unique, and there is nothing to prevent otherwise identical data items being added to the dataspace.
- There is an explicit update event/operation, rather than requiring a combination of “in” and “out”.
- These updates can be identified as reliable or unreliable, and in the latter case may be discarded by the networking or data service infrastructure in the face of congestion or buffer overflow.

These features have been chosen to provide reasonable performance, and to accommodate asynchronous communication (e.g. over lower bandwidth or intermittently connected networks) without penalising local performance. In particular, none of these primitives require remote coordination and so can be performed without the introduction of network round-trip delays (global “in” and “collect” operations cannot avoid network coordination).

KEY EQUIP MECHANISMS

Having introduced the overall structure of EQUIP and the data service this section focuses on a number of key technical elements underpinning EQUIP. In particular: we present its support for pattern matching; the ways in which it can be used to realise distributed computation; and its support for dynamic discovery of devices and services.

Pattern Matching across different languages

EQUIP’s IDL compiler auto-generates pattern matching operations for all IDL-defined classes. Having a general pattern matching operation available for all (IDL-defined) classes provides significant flexibility in using EQUIP. As we have seen the data sharing service relies heavily on this facility. While reflective languages – such as Java – can potentially support pattern matching through the reflection interface, this is a relatively inefficient approach, and does not generalise to C++.

EQUIP’s pattern matching operation determines whether a partially specified or “template” instance matches another IDL-defined object. A template matches an instance if and only if:

- It is of the same class, or a super-class of the instance;
- All fields of primitive types have exactly the same value(s) (e.g. integers, booleans, floats);
- All string type fields in the template are either null exactly equal to the corresponding field on the instance (i.e. null string fields act as wildcards);
- All object reference type fields in the template are either null, or the referenced object – treated as a template – also match the corresponding object reference field in the instance (i.e. null object reference fields act as wildcards); and
- All array type fields in the template are either of length zero, or match element-by-element with the corresponding field in the instance (i.e. zero length arrays act as wildcards).

This definition of pattern matching is quite general, however it does place some constraints on the class designer. In particular, any primitive typed fields that must be wildcarded in any pattern matching operation must be boxed in a separate class which then becomes an object reference field. When set to null in the template this then acts as a wild card.

If the auto-generated pattern-matching operation does not provide the required semantics then the module developer can also over-ride it in their implementation class, to provide classes with different pattern matching semantics. For example, the EQUATOR IRC is also interested in notions of matching that account for elements of context (of whatever form), which are likely to require different forms of pattern matching.

Code Mobility and Distributed Computation

Our descriptions so far have focused on EQUIP’s core data (event and state) sharing facilities. In some applications it is sufficient to share completely passive data, e.g. the state of a button or other input device. However, in other applications it is desirable to also distribute elements of computation. Consider two examples from Collaborative Virtual Environments: dead reckoning, and renderable objects.

Dead Reckoning

Dead reckoning [5] is a technique whereby communication can be traded off against distributed computation. Consider a data item that represents the position of a virtual (or real) vehicle. With purely passive data the item’s value (representing the vehicle’s position) must be updated whenever the vehicle moves. However it is also possible to approximate the vehicle’s movement, e.g. by an initial position, velocity and acceleration. This information can be used to extrapolate the vehicle’s

changing position over time. The data item (now representing the vehicles expected trajectory) can then be updated only when the current extrapolation starts to deviate significantly from the vehicle's actual movement. However each client wishing to determine the vehicle's position must now be able to evaluate the dead reckoning function in use for that vehicle (a distributed or replicated computation).

Renderable objects

3D graphical objects can be distributed as passive data, such as a scene graph, provided that the receiving clients already contain the functionality for rendering that particular scene graph type. However some objects are hard to represent efficiently using a scene graph, for example a continuously deforming surface that represents the time-based evaluation of a mathematical function. Similarly, there are many alternative scene graphs and similar representations of renderable objects, and it is unlikely that a given renderer will include them all. In these situations we would like to share both the code required to render the object and the – hopefully more concise – representation of the object's state for this particular rendering, so that we can ask the object to draw itself (see for example [13]).

This kind of distribution of computation is possible in EQUIP because the data items that are shared in the data space are fully-fledged objects within the host programming language (C++ or Java). While the IDL compiler generates default implementations, the module programmer can provide their own method implementation, e.g. a “draw” method for renderable objects. Because data items are replicated using dynamic code loading the implementation code of the object is loaded into each client and the server, and so can provide local or distributed computation specific to that object class.

Dynamic Discovery

One of the significant challenges of creating interactive systems which span multiple machines is locating and configuring the various elements that comprise the whole system. Service discovery mechanisms can ease this burden by reducing the amount of pre-configuration required, for example to find data servers or other devices and applications automatically. EQUIP currently supports a number of elements of dynamic service discovery, which are described in this section.

In general, EQUIP dataspace provide a flexible context for service advertising and discovery through the publishing and location of data items representing services within the dataspace. However, a prior problem is to locate the particular dataspace to be used for this form of discovery. To address this problem we have expanded the EQUIP URL scheme and the simple name service (mentioned already) to support multicast-based queries of the form “`equipm://multicastgroup:port/name`”.

By specifying a standard/default multicast group and discovery dataspace name EQUIP applications can easily locate the current network-local dataspace service. This is comparable to JINI's location of service directories [25], although our implementation does not currently support multiple redundant name services or discovery dataspace on a single network.

Once a potential server has located the local discovery dataspace it can then use standard data items (from the module `equip_service`) to advertise its existence. Alternatively, at least for sensors and input devices, it can simply begin to create and maintain data items in the discovery dataspace that reflect its current state.

Clients can then use the normal item monitor facilities to be informed of the existence of items that indicate the existence of required services (or the actual data in the case of the sensors/input devices mentioned above).

EQUIP IN USE

Equip has been used to realise a number of different interactive experiences that combine the real and the virtual. In developing these different arrangements we have created a number of input and output modules within the EQUIP framework, which currently include:

- Publishing input from various devices, including GPS receiver, tilt sensor, mouse/keyboard, PC joystick, 2D GUI input simulator (e.g. to simulate a joystick on a handheld device), RF-ID tag reader and video-based light tracking.
- Interfaces to the MASSIVE-3 CVE [12] system, e.g. to monitor and create 3D virtual objects, and to create and control 3D graphical views.
- Various 3D rendering objects, including simple solid geometries, abstract data visualisations, embedded views of MASSIVE-3 worlds, and geo-referenced objects and positions (e.g. globally located using GPS).
- Audio control clients, for the MASSIVE audio service, and to generate high-level MIDI messages from EQUIP data.

To date these have been used in a number of projects and demonstrations, including the Augurscope tripod-based mixed reality interface [21], various collaborative experiences and experiments between virtual tourists in UCL's immersive virtual reality ReaCTor display and physical visitors with handheld wireless devices [22], and the Unearthing virtual history experience developed as part of the shape Museum demonstrator, described next.

The SHAPE Museum Demonstrator

In the SHAPE museum demonstrator [3] EQUIP performs an integration role, connecting the physical world – in the form of various input and output devices embedded or mobile within it – to a corresponding virtual world, realised in the MASSIVE-3 CVE system [12]. This virtual

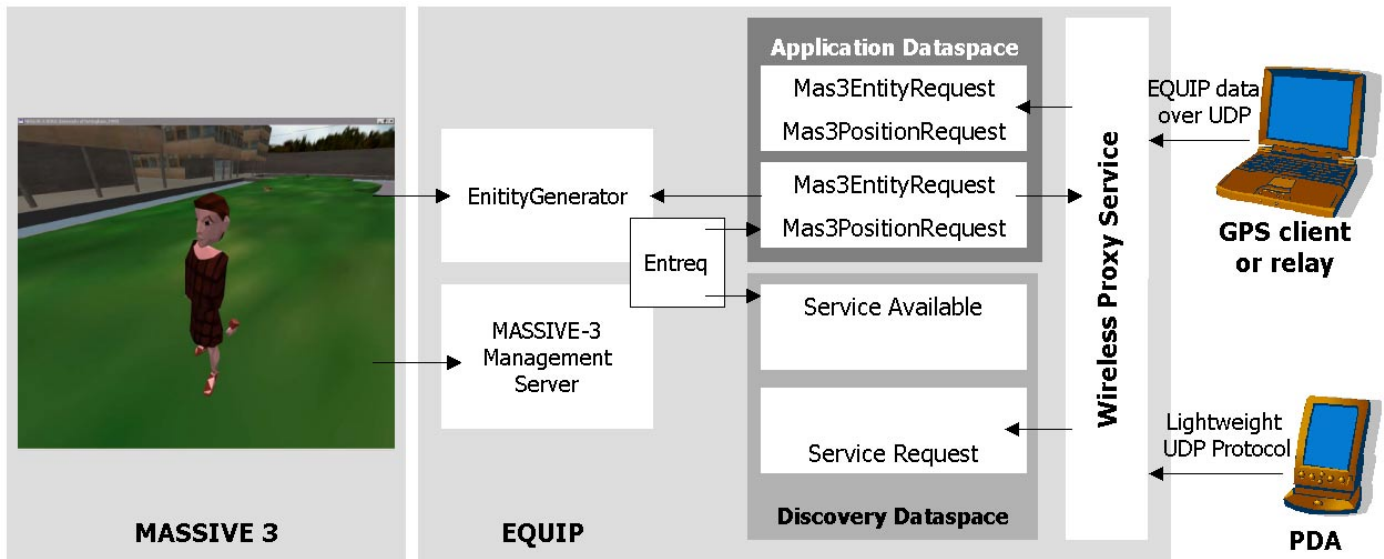


Figure 7. Supporting the wireless user in the SHAPE museum demonstrator

world can be experienced by those in the real world through interfaces on a handheld device (a Compaq iPAQ), through an audio landscape (presented via a Laptop PC) and through a special purpose display/projection device called the periscope. Each of these devices present different experiences of the shared environment.

The MASSIVE-3 CVE system uses its own data distribution facilities to construct and maintain a shared 3D audio-graphical virtual world. Two standard EQUIP components provide a general-purpose bridge between EQUIP and MASSIVE-3. These are:

- A 'Management server', that can create and render a 3D graphical view of a MASSIVE-3 virtual world. It also conveys the user's mouse and keyboard input to EQUIP, and can relay information about avatars and objects in the virtual world to EQUIP.
- An 'Entity Generator', that can create and animate 3D virtual objects in MASSIVE-3 worlds according to data in an EQUIP dataspace.

Figure 7 shows how these components are used in the SHAPE museum demonstrator in order to:

- give the outdoor wireless user an avatar (virtual body) within the MASSIVE-3 virtual world (from a GPS client to the Entity Generator); and
- give the outdoor wireless user information about objects in the virtual world such as their proximity (from the management server to a simple 2D client running on the PDA).

Access for intermittently connected wireless devices is provided through a proxy service. This arrangement

allows clients to send and receive EQUIP data items using the connectionless UDP transport protocol (which is not yet supported for native EQUIP communication – see future work). If the proxy receives an EQUIP item that does not appear within the application dataspace it will add the item on behalf of the client. Further updates can also be issued through the proxy, as and when relevant data items arrive from the client. When experimenting with this arrangement we found that low memory devices could not receive and unmarshal large numbers of events at a reasonable rate. As an interim measure an additional lightweight ASCII protocol was developed to communicate between the proxy and low resource devices. However, we have subsequently run full EQUIP clients on small devices of this kind, and believe that careful thread and communication management will allow us to avoid the need for additional proxy protocols in most applications.

RELATED WORK

The domain of interest – distributed interactive systems – may be viewed as one approach to ubiquitous computing, which is often characterized in terms of environments that place digital information in physical spaces [28]. The supporting infrastructure often exploits the contextual information provided by the space to manage the interaction between devices, services and users [23]. Similarly, the development of virtual environments and the construction of collaborative virtual environments (CVEs) have used virtual spaces to manage interaction.

The majority of supporting services and protocols for this kind of ubiquitous computing have focused on representing and accessing services and the provision of communication mechanisms between devices. This has included the extension of protocols used within World

Wide Web [30] and a number of general device discovery mechanisms [27], [25].

Each of these different platforms often embody their own philosophy and approach, and place fairly significant demands on devices that wish to link with a supporting infrastructure. In contrast we wish to focus on a low cost lightweight infrastructure that allows a number of devices to share information. Although existing platforms focus on communication between devices some of these protocols already incorporate certain facilities for sharing. For example, Jini incorporates JavaSpaces [24]. A similar model of shared state has been exploited within the GUIDE system [4]. Similar shared state based models exist in collaborative virtual environments such as Dive[26], Massive[12] and Spline[2].

In the development of EQUIP we have focused on allowing a number of devices to communicate across the boundary between real and virtual, thus permitting the convergence of these worlds. The EQUIP data sharing service supports the sharing of arbitrary data among heterogeneous distributed applications, embracing applications written in both C++ and Java.

The EQUIP platform extends work on tuple spaces (e.g. LINDA [10], Tspaces [14], Limbo [6]) and shared virtual environments to provide an active shared state infrastructure that is equally accessible from large virtual environments and small handheld devices. As well as application state, the platform also allows service information to be shared, enabling a number of flexible context driven resource discovery policies to coexist.

In EQUIP all communication is performed indirectly, via the data space, rather than through direct inter-application communication. This same kind of indirect communication is seen in other pattern-driven distribution systems, such as tuple spaces and generic event distribution systems such as Elvin [20]. This indirection facilitates the construction of loosely coupled distributed systems that are needed to bridge between the physical and digital.

This is in contrast to the more direct patterns of communication found in the Context Toolkit [7]. In Dey's terms Context Toolkit adopts a real world/widget view of context and devices, whereas EQUIP is much closer to the blackboard paradigm [7, p68,69]. Dey observes that the blackboard approach frees application developers from focussing on specific device details, and supports a more declarative style of programming. On the other hand it is a more challenging paradigm within which to integrate history, access control and privacy considerations. However we believe that extensions to the blackboard paradigm will permit support for these issues (e.g. content annotations, explicit representations of time and history), although these challenges have yet to be addressed within EQUIP.

CONCLUSIONS AND FUTURE WORK

While we are satisfied with current implementation there are still many areas in which we plan to enhance it. Some of these are identified below.

The current process of building and installing EQUIP relatively difficult, and we intend to create a much simpler bootstrapping installation, perhaps using a trusted Java applet. The current system also requires compatible versions of all EQUIP modules to be installed on all machines in advance. We intend to extend this to support dynamic downloading and updating of EQUIP modules, simplifying ongoing maintenance.

While the current system has reasonable support for service discovery, this addresses only part of the configuration and management process of a multi-application multi-machine interface. An immediate priority is to provide stronger support for the complete activity of application deployment, configuration and management.

We have architected the data sharing service to cope with the characteristics of wireless networks, e.g. by emphasising local operations, asynchronous communication, and relatively independent dataspace (partial) replicas at each client. However, there is further work to do in comprehensively supporting this mode of use. In particular, the current low-level networking employs TCP, which imposes constraints on ordering and network timeouts that are less than ideal. The current implementation also does not support dataspace associations (client to server) that can outlive periods of disconnection. Currently, all replicated items will be removed when the network connection fails, and will only be restored when the network connection is restored. This needs to be supported to allow meaningful activities to continue on wireless devices during periods of disconnection.

The current system has no explicit notion of history or previous state. We have started to apply the notions of temporal links and record and replay (from MASSIVE-3 [11]) to EQUIP, as one approach to preserving and accessing historical information. It is likely that other facilities will also be required to support general notions of historical context.

ACKNOWLEDGEMENTS

This work has been carried out within the EQUATOR Interdisciplinary Research Collaboration funded by EPSRC in the UK and the SHAPE project under the European V Framework Disappearing Computer Initiative.

REFERENCES

1. Bamboo, <http://watsen.net/Bamboo> (verified 5 April 2002).
2. Barrus, J.W., Waters, R.C., Anderson, D.B.: Locales: Supporting Lange Multiuser Virtual Environments, in IEEE Computer Graphics and Applications (1996) 50-57

3. Steve Benford, John Bowers, Paul Chandler, Luigina Ciolfi, Martin Flintham, Mike Fraser, Chris Greenhalgh, Tony Hall, Sten Olof Hellström, Shahram Izadi, Tom Rodden, Holger Schnädelbach, and Ian Taylor, "Unearthing virtual history: using diverse interfaces to reveal hidden virtual worlds". In *Proc Ubicomp 2001*, pages 1-6. ACM, November 2001.
4. Cheverst, K., Davies, N., Mitchell, K., Friday, A.: *The Role of Connectivity in Supporting Context-Sensitive Applications*, Lecture Notes in Computer Science, No. 1707, Springer-Verlag, Heidelberg, Germany (1999) 193-207
5. Cooke, J.C., Zyda, M.J., Pratt, D.R. and McGhee, R.B., "NPSNET: Flight Simulation Dynamic Modeling using Quaternions," *PRESENCE: Teleoperations and Virtual Environments*, vol. 1 no. 4, Fall 1992, pp. 404-420.
6. Davies, N. , Wade, S.P., Friday, A., Blair, G.S.: "Limbo: A Tuple Space Based Platform for Adaptive Mobile Applications", *Proc. Int. Conference on Open Distributed Processing/Distributed Platforms (ICODP/ICDP '97)*, Toronto, Canada (1997) 291-302
7. Anind K. Dey, "Providing Architectural Support for Building Context-Aware Applications", PhD Thesis, Dept. of Computer Science, Georgia Institute of Technology, November 2000.
8. EQUATOR IRC, <http://www.equator.ac.uk> (verified 5 April 2002)
9. EQUIP, <http://www.equator.ac.uk/technology/equip> (verified 5 April 2002).
10. David Gelernter, "Generative Communication in Linda". *ACM Transactions on Programming Languages and Systems*, Jan. 1985, Vol. 7, No. 1, pp. 80-112.
11. Chris Greenhalgh, Jim Purbrick, Steve Benford, Mike Craven, Adam Drozd, Ian Taylor, "Temporal links: recording and replaying virtual environments", in *Proceedings of the 8th ACM international conference on Multimedia, 2000* , Marina del Rey, California, United States, 67 – 74, ACM Press, New York.
12. Greenhalgh, C., Purbrick J., Snowdon, D.: *Inside MASSIVE-3: Flexible Support for Data Consistency and World Structuring*, *Proc. of ACM Conference on Collaborative Virtual Environments (CVE2000)*, San Francisco, USA (2000) 119-127
13. R. Hubbold, J. Cook, M. Keates, S. Gibson, T. Howard, A. Murta, A. West, and S. Pettifer. "GNU/MAVERIK : A micro-kernel for large-scale virtual environments". *Presence: Teleoperators and Virtual Environments*, 10:22-34, February 2001. ISSN 1054-7460.
14. IBM Corp.: TSpaces Technology, <http://www.almaden.ibm.com/cs/TSpaces/>
15. Ian MacColl, Barry Brown, Steve Benford, Matthew Chalmers, Ruth Conroy, Nick Dalton, Areti Galani, Chris Greenhalgh, Danius Michaelides, Dave Millard, Cliff Randell, Anthony Steed, Tom Rodden, Ian Taylor, Mark Weal, "Shared Visiting in EQUATOR City", submitted to *Pervasive 2002*.
16. Milgram, P., Kishino, F. A taxonomy of mixed reality visual displays, *IEICE Transactions on Networked Reality*, E77D (12), 1321-1329, 1994.
17. Netscape Portable Runtime, NSPR, <http://www.mozilla.org/projects/nspr/> (verified 5 April 2002).
18. OMG, "CORBA/IIOP Specification", http://www.omg.org/technology/documents/formal/corba_iiop.htm (verified 5 April 2002), esp. chapters 3 and 5.
19. Rogers, Y., Scaife, M., Harris, E., Phelps, T., Price, S., Smith, H., Muller, H., Randall, C., Moss, A., Taylor, I., Stanton, D., O'Malley, C., Corke, G. & Gabrielli, S. (2002) Things aren't what they seem to be: innovation through technology inspiration. To appear in *DIS2002 Designing Interactive Systems Conference*, ACM. London, 25-28th June.
20. Bill Segall and David Arnold, "Elvin has left the building: A publish/subscribe notification service with quenching", *Proceedings AUUG97*, Brisbane, Australia, September 1997.
21. Holger Schnädelbach, Boriana Koleva, Martin Flintham, Mike Fraser, and Paul Chandler, "The Augurscope: A Mixed Reality Interface for Outdoors". In *Proc. CHI 2002*, pages 1-8. ACM Press, April 2001.
22. Anthony Steed, Steve Benford, Nick Dalton, Chris Greenhalgh, Ian MacColl, Cliff Randell, Holger Schnädelbach, "Mixed-Reality Interfaces to Immersive Projection Systems", *7th annual Immersive Projection Technology Symposium*, March 24-25, 2002, Orlando, FL.
23. Streitz, N.A., Geißler, J., Holmer, T., Konomi, S., Müller-Tomfelde, C., Reischl, W., Rexroth, P., Seitz, P., Steinmetz, R.: *i-LAND: An interactive Landscape for Creativity and Innovation*, *Proc. ACM Conference on Human Factors in Computing Systems (CHI '99)* , Pennsylvania, USA, (1999) 120-127
24. Sun Microsystems Inc.: JavaSpaces Technology, <http://java.sun.com/products/javaspaces>
25. Sun Microsystems Inc.: Jini connection technology, <http://www.sun.com/jini/>
26. Swedish Institute of Computer Science (SICS): DIVE - A Toolkit for Distributed VR Applications, <http://www.sics.se/dce/dive>
27. Universal Plug and Play Forum: Universal Plug and Play (UPnP), <http://www.upnp.org/>
28. Want, R., Schilit, B.N., Adams, N.I., Gold, R., Petersen, K., Goldberg, D., Ellis, J.R., Weiser, M.: An overview of the ParcTab ubiquitous computing experiment. *IEEE Personal Communications Magazine* (1995) 28-43
29. K. Watsen and M. Zyda, "Bamboo - A Portable System for Dynamically Extensible, Real-time, Networked, Virtual Environments," presented at 1998 IEEE Virtual Reality Annual International Symposium (VRAIS'98), Atlanta, Georgia, 1998.
30. World Wide Web Consortium.: eXtensible Markup Language, <http://www.w3.org/XML/>