The Event Heap: A Coordination Infrastructure for Interactive Workspaces

Brad Johanson and Armando Fox
Stanford University
Gates 3B-376
Serra Mall
Stanford, CA 94305-9035
bjohanso@graphics.stanford.edu, fox@cs.stanford.edu

Abstract

Coordinating the interactions of applications running on the diversity of both mobile and embedded devices that will be common in ubiquitous computing environments is still a difficult and not completely solved problem. We look at one such environment, an interactive workspace, where groups come together to collaborate on solving problems. Such a space will contain a heterogeneous collection of both new and legacy applications and devices. We propose the Event Heap, a coordination model most similar to tuplespaces, as being appropriate for such environments. We also present a prototype implementation of the Event Heap, and show that the system has performed well in actual use over the last two years in our prototype interactive workspace, the iRoom.

1 Introduction

Improvements in device technologies and falling costs are rapidly enabling the original vision of ubiquitous computing [26]. Devices from large wall-sized displays to small PDAs can easily (and wirelessly) be networked together in localized areas, forming the hardware side of the ubiquitous computing environment. Once connected together, however, the problem becomes how to allow software programs running on the devices to coordinate with one another in a flexible and intuitive manner. Such devices do not generally integrate easily, either with one another or with existing software, unless they were designed to do so a priori.

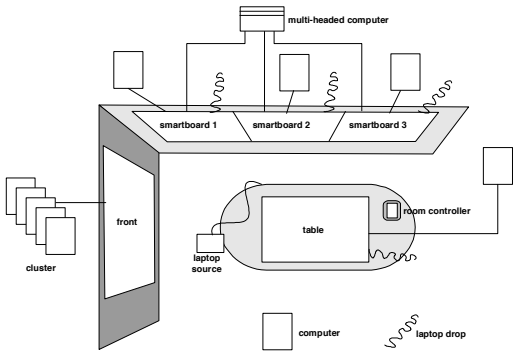
Many programming models and systems have been proposed for this type of coordination in ubiquitous computing scenarios. Based on our experience with a prototype room-based ubiquitous computing environment (interactive workspace), the realities of this type of environment make the existing models incomplete or inadequate. Interactive workspaces systems must be able to tolerate a dynamic environment, as mobile devices come and go, as well as maintain a high degree of robustness and availability despite inevitable software and hardware failures. Further, both because this research area is still young and growing rapidly, and because of the implicit *ad-hoc* nature of ubiquitous computing interactions, it is important that any coordination model allow the rapid integration of new devices and systems.

Our own project, Interactive Workspaces, investigates the systems and HCI issues that arise in these technology rich spaces. People gather in these environments to do naturally collaborative activities such as design review, and brainstorming. Compared to other projects, which we review more thoroughly toward the end of the paper, we are focusing on providing infrastructure for dynamic, heterogeneous and ad hoc collections of devices, applications and operating systems, all of which may be either new or legacy.

From our observations of usage and application development in our prototype interactive workspace, the iRoom, we determined that a coordination infrastructure for such a space must be:

- Applicable to many different types of ubiquitous computing applications.
- Able to support portability of applications across installations.
- Friendly to existing languages and environments, and portable to new ones, in order to make it straightforward to support a wide range of devices and leverage their existing application bases.
- Robust to transient failures, so that experimentation with new devices does not destabilize an existing system.

In light of the above observations and requirements, we make the following contributions:



(a) iRoom Layout

(b) Construction Management in the iRoom

Fig. 1. The iRoom

First, we identify the need for a general-purpose coordination system, in the spirit of [10], for interactive workspaces. Further, we propose that the Event Heap, a model derived from tuplespaces, is suitable for such a system, and show why we found tuplespaces to be a good starting point.

Next, we explain how the Event Heap differs from the basic tuplespace model that, and why we found these changes necessary in an interactive workspace. Specifically, the changes are the addition of flexible, typed, self-describing tuples, tuple sequencing, tuple expiration, default mechanisms for tuple routing, and query registration.

Finally, we present our implementation of the Event Heap and discuss our experience with applications using the infrastructure across a wide variety of devices and software platforms over the past two years.

1.1 Interactive Workspaces

An *interactive workspace* is a localized ubiquitous computing environment where people come together for collaborations. To explore this space, we built a prototype interactive workspace, the iRoom, which features three rear projected SMART Board touch screens [22] along one wall, a bottom projected table, and a custom 12 projector tiled front display driven by a workstation cluster. Except for the front display, all of the machines driving displays are Windows machines in order to facilitate running legacy applications. In addition, the room has wireless LAN coverage, which allows laptops or PDA's to communicate with the other machines in the room. Figure 1a shows the layout of the iRoom.

We now provide a scenario set in the iRoom that reflects how research groups collaborating with us hope to use interactive workspaces (a video version is linked on the web from: http://iwork.stanford.edu).

Consider a group of construction management engineers and contractors using the iRoom to plan a major construction project. (We are working with the civil engineering department on just such a project [15], but similar scenarios apply for many domains requiring multi-person collaborations and interaction with large amounts of data.) Upon entering the workspace, one group member uses a touch sensitive tablet at the room entrance to turn on the lights and the three projectors for the touch screens on the side wall.

They begin the meeting by following a web-page outline the meeting leader has prepared and brought in on his laptop which he displays on the left most touch screen. Each topic is a hyperlink that brings up related data for that topic on the other displays in the room. Some of that data is in the form of web pages, while other data is brought up in specific construction site modeling and planning applications, some of which were not originally designed to run in the iRoom. Figure 1b shows a photograph of the iRoom in use for a prototype of such a scenario.

Later in the meeting, it becomes clear that there is a problem with completing part of the construction on schedule. They bring up a top down map of the construction site on the table, a 3D model of the construction site which shows the project state for any given date on one touch screen, some financial information on another touch screen, and the project scheduling software on the third. All of these are separate stand-alone applications, but the data being displayed across the applications is automatically associated. Thus when the users select or make changes in one view, the other views immediately reflect the new state. When they have solved the problem and the meeting is over, the users store the updates on their laptops, and shut down the

room using a simple web based room control page they load on their laptop using the wireless network in the room.

The goal of the coordination infrastructure is to facilitate the kind of fluid application association and linking, and the multiple pathways of control as presented in this scenario.

The rest of the paper proceeds as follows. We first discuss coordination languages, and our choice of the tuplespace model as a starting point for the Event Heap. We next present the extensions to tuplespaces we made to create the Event Heap, our prototype implementation, and applications implemented using the prototype. We conclude with some discussion and a review of similar projects.

2 A Tuplespace Coordination Model For Interactive Workspaces

We assert that most room wide applications will consist of traditional applications and devices composed into an ensemble. The problem is therefore to determine which coordination model best facilitates these types of composition such that the user has the impression of using one distributed application.

2.1 Coordination-Based Programming Background

In [10], Gelernter and Carriero proposed that computation languages and coordination languages should be thought of as orthogonal, with computation languages expressing how calculations proceed, and coordination languages expressing the interaction between autonomous processes. They assert that providing a coordination mechanism separate from the computational language provides two key features: *portability*, by providing a computation language independent mechanism of coordination, and support for *heterogeneity* by allowing devices and applications to coordinate with one another even if they are based on different hardware or languages. Further, providing a general-purpose coordination language, as opposed to many specialized ones, is *economical,* because programmers need learn only one coordination language, and provides *flexibility,* since it can be used to express any style of coordination. They propose that Linda [1], which is based on the tuplespace model, is one such general purpose coordination language, and therefore has these features.

In the tuplespace model all participants coordinate through a commonly accessible tuplespace. Tuples, which are a collection of ordered type-value fields, may be posted to the space, or read from the space in either a destructive or non-destructive manner. The tuple to be retrieved is chosen by a template tuple specified by the retrieving application. The template contains precise values for the fields to be matched, and wild cards for fields containing data to be retrieved. This system is simple, can be shown to be general, and has flexible matching semantics.

2.2 Tuplespace Advantages for Interactive Workspaces

The features of tuplespaces advocated by Gelernter and Carriero—portability, heterogeneity, economy and flexibility—are the same as ones which are important for an interactive workspaces coordination model. This makes the tuplespace model a good starting point for an infrastructure in this domain. Other characteristics of the tuplespace model that are important for interactive workspace coordination are simplicity, and good failure isolation and tolerance.

Since tuplespaces are simple and flexible they are easy to deploy on many devices and platforms. There are only three primitives, making it simple to port to new platforms, add support to an existing application, or write a wrapper for an application with a programmatic interface. Coordination state is stored in the infrastructure instead of in individual clients which makes client code small and straightforward to implement even for impoverished devices. Since tuplespaces are general-purpose, other coordination types, such as RPC, can be implemented on top of it if they are more appropriate for some task.

Tuplespaces also support easy coordination among multiple applications, including the ability to adapt applications not originally designed to work together. Multicast communication between disparate groups of devices and applications is easy since multiple applications can get a copy of the same tuple if they all match for it. Tuplespaces also inherently provide the following features:

- **Anonymous communication:** There is no need to explicitly rendezvous applications—as long as two applications understand the same event types they will automatically coordinate with each other. Users can bring up applications on the display they want in an interactive workspace and the applications should coordinate correctly.
- **Interposability:** Since tuples are public and indirectly sent between applications, an intermediary can pick up a tuple from a source and put back one or more tuples of different types which will cause the appropriate action in a receiver or receivers. This allows applications not originally intended to work together to coordinate.

- **Snooping:** The tuplespace model allows one component to snoop on tuples being sent among other components without impinging on their behavior. Information in that tuple can then be used to affect the local behavior of the snooping application.

The indirect interaction mechanism of the tuplespace discourages strongly interdependent applications, which helps with failure isolation. As long as the tuplespace infrastructure can tolerate failure in clients, a client should not cause others to fail. Tuples also persist, decoupling applications in time as well as space. This allows applications to retrieve communications that were sent before they were running, or as they were restarting after a crash.

2.3 Adapting Tuplespaces for Interactive Workspaces

While the tuplespace model is a good general-purpose system for coordination, we found that certain extensions were necessary for the interactive workspaces domain. It should be noted that while some of these extensions have been implemented in other systems which extend the tuplespace model, we are aware of no single system that incorporates the specific set which we have found necessary for this domain. The extensions are:

Flexible, typed, self-describing tuples: In an interactive workspace, applications may not have been designed to work together. To allow developers to infer meaning of tuples by spying on application communication, it is important to make fields self-describing in tuples. This means that every field needs a name in addition to the type and value. For extensibility, it is desirable to permit applications to add extra fields to tuples without breaking the matching semantics of older applications. Thus, we make extra fields allowable, and field-order irrelevant to matching. Finally, there is a need to disambiguate the semantic meaning of tuples, so a field that indicates the type of the tuple is included. This type also implies some minimal set of fields that will be present, and allows the differentiation of events with fields of the same name but different intent. For example, an "ItemSelected" field may be present in both a database update event and a 3d-model selection event. Tuple typing is used in Javaspaces [21], L2imbo [9], and other systems. TSpaces [27] allows self-describing fields. Although some object oriented tuplespace implementations support matching by subclass, as far as we know no tuplespace system supports allowing templates to match tuples that are a superset of themselves.

Tuple Sequencing: Traditionally, if multiple tuples exist that match the template tuple on a retrieve operation, any of the matching tuples can be returned. Tuple sequencing means that receivers always get the earliest matching tuple they haven't yet seen. Sequencing insures that applications requesting state change tuples will get tuples exactly once, and in order, rather than fetching the same tuple repeatedly. Since applications may sometimes want to peek at tuples, a 'snoop' method is needed to return copies of all matching tuples without effecting sequencing. Sequencing has also been found useful when applying the tuplespace model to other domains. See, for example, [14]and [19].

Expiration of Tuples: Since sources and receivers are decoupled, a source need not have a corresponding running receiver. This may cause emitted tuples to build up in the tuplespace—for a real world system this means that the performance steadily decreases. To ameliorate this problem, all tuples are given a 'TimeToLive' field that specifies how long they will persist in the tuplespace before being "garbage collected." The expiration also facilitates human time-scale inter-application coordination by preventing action upon a submitted tuple from occurring long after the triggering event. For example, a light should turn on within a few seconds, or not at all—turning on hours or days later when some key component comes back on line is not acceptable. Tuple expiration is also implemented in the TSpaces system [27].

Default Routing Fields: While anonymous communication tends to minimize the interdependency between applications, it makes it more difficult to route information between the appropriate parties. To allow targeting of transmitted tuples to specific applications or groups of applications, a standard set of routing fields are provided which are filled out automatically by the tuplespace infrastructure. For example, the infrastructure automatically sets a source application field to indicate who created the tuple, and automatically sets template tuples to match tuples with a destination application field set to match them. Details of the implementation of this are in Section 3.2. While it is possible for an application programmer to create a standard set of fields for routing for his own application, only by specifying the fields at the infrastructure level can interoperability be assured among applications written by different authors. Further, having the fields created and set automatically makes it more likely that they will be used, since authors do not have to write routing code themselves.

Query Registration: The standard tuplespace model requires polling in order to retrieve tuples. This introduces the problem that tuples which expire or are destructively removed between successive polls by an application will be missed. In the cases of applications that are snooping, debugger applications, and logger applications, this is a serious problem. To alleviate this, the Event Heap model allows applications to register template tuples with the system. As long as they remain registered, a callback is made to the application every

time a matching tuple is placed. This mechanism is similar to the primary message dissemination method in publish-subscribe systems. TSpaces [27] and LIME [16] both support similar registration methods.

For an interactive workspace, users need to be able to dynamically compose the application components they are using into an ensemble. This differs from the original intended use of tuplespaces which was to construct a set of applications designed from the ground up to act as an ensemble. In our case, the programmer doesn't know in advance with which other applications their component will be coordinating. By adding flexible, typed, self-describing events, and using intermediation and snooping, the tuplespace model is adapted to help support this sort of dynamic composition.

2.4 Design Alternatives

The tradeoffs between tuplespaces and other coordination mechanisms are well known, and one of our main contributions is to identify tuplespaces as well suited to interactive workspaces. We also considered publish-subscribe, RMI/RPC, and message passing systems.

Publish-subscribe provides many of the same advantages as tuplespaces. The main difference is that messages in publish-subscribe systems have no persistence, so there is no inherent way for restarting applications to pick up recent messages. This makes it more difficult to keep things running through a failure.

Both RMI/RPC and message passing suffer from drawbacks similar to one another in the interactive workspaces domain. Like publish-subscribe, there is no temporal persistence to coordination. In RMI/RPC, language independence is more difficult since different languages have differing method call protocols. Since communication is direct, getting programs not designed to work with each other to rendezvous is more difficult, and non-point-to-point communication is more difficult. In particular snooping and intermediation are not well facilitated by the basic coordination model. Finally, since applications need to be aware of the other application to which they are communicating, some additional rendezvous mechanism must be provided by any infrastructure based on RMI/RPC or message passing.

3 The Event Heap Implementation

Although early versions of our prototype implementation of the Event Heap were built on top of a tuplespace implementation from IBM called TSpaces from IBM [27], our current version is stand alone. The system is client-server based with tuplespace state stored on the server machine. While the server is a single point of failure, individual Event Heap client applications automatically reconnect if the server goes down and is then restarted. This combined with a dedicated web server that handles requests to restart the server minimizes the problems with server failure.

Since the semantics as described in section 2.3 are slightly different from a standard tuplespace, in the Event Heap tuples are called *events*. This reflects their intended use as a means of notifying other applications in the workspace of an occurrence, or of requesting that other applications update their state or perform some task. In the sequel the term *event* will be used when referring to tuples in the Event Heap, and *tuple* will be used to refer to the standard Linda-style tuple.

Although under most normal loads the current implementation has a latency of less than 20 ms, the current version of the Event Heap is used primarily for coarse, high-latency coordination between applications running in the iRoom.

3.1 Event Format Description

The basic event used by the Event Heap is a tuple with certain mandatory fields. As mentioned in section 2.3, flexible typing provides several advantages in the interactive workspace domain, so we ignore field order and tuple size in performing matching. This means that fields are always referred to by name and type rather than their index in the tuple. The fields include the 'EventType' field which, as the name implies, is used to indicate the type of the event, a 'TimeToLive' field, which determines expiration, some fields for routing, and some internal fields that aid in sequencing the events. All fields except for 'TimeToLive' and the internal use only fields are string type fields. This makes events relatively easy to parse when looking at them using the Event Heap debugger or debug traces.

3.2 Event Retrieval and Routing

The Event Heap provides additional operations to retrieve events beyond the basic destructive and non-destructive read operations in the standard tuplespace model. There are non-blocking versions of the basic calls, and all of the calls will accept an array of template events and return an event that matches one or more of these. There is a 'snoopEvents' call which retrieves events without effecting sequencing. Finally, events can be retrieved using query registration, one of the aforementioned extensions.

As mentioned in section 2.3, one of the desired additional capabilities for a tuplespace-based system in an Interactive Workspaces is some means of routing tuples. The Event Heap accomplishes this by providing standard source and target fields which allow routing to or from individual objects using the Event Heap, applications, devices, people, or groups.

This works as follow: when events are posted, the source version of each field is automatically set by the Event Heap client code. For example, the 'SourceApplication' field is set to the name of the application. Receivers can then match on certain application names in that field to receive events from that application. Further, when an application retrieves using a template, the target version of each field is also automatically set. The 'TargetPerson' field, for example, might get set to 'Bob.' Now sources can set the 'TargetPerson' field to 'Bob' if they want their event only to be picked up by applications currently being used by 'Bob.' If a source sets does nothing to the target fields, they default to being wildcards, which means the event will be picked up by all receivers that match the rest of the fields correctly. This system allows flexible and standardized point-to-point, multicast and broadcast communication.

3.3 Event Sequencing

To perform sequencing, each source tags every generated event with a Source, a SessionID, and a SequenceNum (sequence number). The Source needs to be unique among all sources participating in the Event Heap, so a random integer is always appended to the name specified by the calling application.[1] The SessionID is chosen randomly every time a source is instantiated, and is used to differentiate between events from before and after an application restart. SequenceNum starts at one and is incremented for each new event of a given type submitted during a given session.

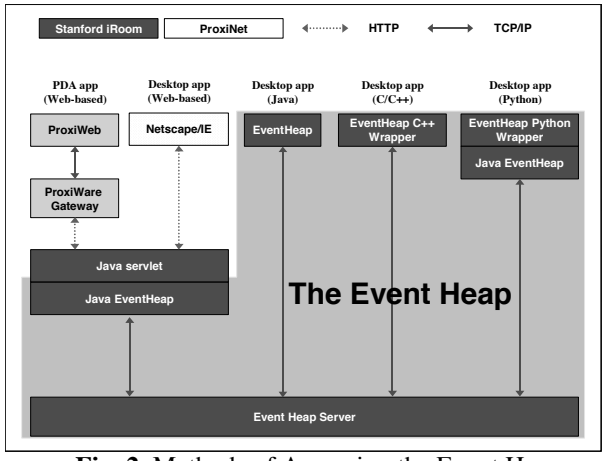


Fig. 2. Methods of Accessing the Event Heap

On the receiver side, sequencing is accomplished by keeping track of the most recent event received from any given source. This information is sent with retrieval requests and the server will only send back events that are newer than the most recent one indicated as seen by the receiver. Keeping the receiver specific state on the receivers insures that it isn't lost if the server gets restarted.

3.4 Integrating Diverse Programming Environments and Devices

A key design goal is supporting a heterogeneous collection of machines and legacy applications. To do so, we have implemented a variety of ways for applications to access the Event Heap as shown in Figure 2. The main client and server implementations of the Event Heap are in Java. The total size of the Event Heap client library in Java is about 45KB, so it is deployable to most devices. There is also a native C++ Event Heap client under Windows, and a Python client which works by wrapping the Java client. We also provide a web pathway that lets users encode event submission in URLs on web pages. This works via HTTP form submission to a Java servlet, and has allowed many basic interactions to be easily prototyped by simply creating a web page with the appropriate event submission URLs. This path has proven useful in allowing PDAs to participate in controlling the iRoom, since currently even Palm-type devices have web browsers available (for example the ProxiWeb [18] browser).

Using the currently available paths and software API's, the Event Heap is currently supported in some form on Windows, Linux, Palm OS, and Windows CE. There is also a well defined TCP/IP protocol for speaking to the Event Heap server, so it is relatively straight-forward to write a client for any new platform that has socket support.

4 Event Heap Applications

In this section we present some applications built on the Event Heap that are in use in the iRoom or are deployed in other interactive workspaces that use the Event Heap. Ten to twenty applications have been written which use the Event Heap since we deployed the first version. In this section we share some exemplary applications—Multibrowsing, SmartPresenter and the CIFE Suite—that demonstrate how the Event Heap is able to facilitate coordination and provides the desirable features we outlined in the introduction.

4.1 Multibrowsing

Multibrowsing [13] is a system that allows one to call up web pages or other data on any machine in the iRoom by submitting a multibrowse event. Each machine that is a valid target for multibrowsing runs a multibrowse daemon that waits for events with the 'Target' field set to

[1] If no source name is specified, the application name with a random integer is used instead.

itself, and then executes the command embedded within. Since the daemon uses Windows shell extensions to execute commands, URLs are brought up in the default web browser and any other file based data is opened in the appropriate application.

Using the web path to the Event Heap, users are able to encode requests to pull up web pages, data, or applications on the other machines in the iRoom using links on web pages. We also have a script/plug-in that works with Internet Explorer and allows users to redirect the current page, or the target of a hyperlink to any display in the iRoom using the right-click menu. The same menu allows web pages to be pulled from remote displays by requesting the URL displayed on that page from the remote multibrowse daemon. Finally, there is a Java applet that allows users to drag content from any machine running a web browser to an iconic representation of the displays in the room, causing the information to brought up on that display.

Currently most multibrowse content and applications are designed only for the iRoom, so URLs and other hard-coded triggers for multibrowse event submission are not portable to other environments. Due to the ability to intermediate, however, we were able to construct *mbforward*, a simple application that picks up multibrowse events with a certain values in their 'Target' fields and automatically re-routes them to different machines. Using this mechanism the CIFE group [15] has been able to demonstrate multibrowse scenarios tailored for the iRoom on a set of laptops for demonstrations in other locations.

4.2 SmartPresenter

SmartPresenter is a multi-display, multi-object presentation program for interactive workspaces. While traditional presentation programs coordinate the display of slides across time, SmartPresenter coordinates the display of information across both time and display surfaces. For example, a presentation might specify that at time-step 4, slide 17 from a Power Point presentation be shown on the left touch screen, a 3-d model be displayed on the high-resolution front screen, and web pages be displayed on the other two touch screens.

The presenter application proper is written in Java and can be run anywhere in the iRoom. It reads a stored script that specifies which events are to be sent at any point during the presentation. It waits for presentation control events telling it to advance, step backward, or jump to some specific point in the presentation, and then sends the events appropriate for that point in the presentation.

Each machine with a display in the room runs a viewer daemon which responds to viewer control events by loading the specified information. There is special support for PowerPoint which has been wrapped using Microsoft Office Automation [4], a programmatic interface to control applications in the Microsoft Office suite. The wrapper allows the viewer to explicitly call forward, backward and build commands.

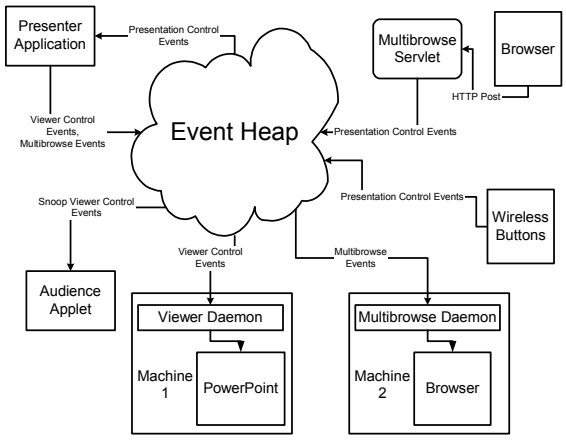


Fig. 3. Application Paths into the Event Heap

We could also easily construct an audience applet which allows users on a laptop to snoop on the presentation control events and display presentation content on their laptop. Figure 3 shows how all the pieces fit together. Note that view control and multibrowse are the only type of event shown, but theoretically any event can be emitted by the presenter application.

The SmartPresenter application demonstrates several of the important features of the Event Heap:

- **Composability:** By creating one presenter application any number of Event Heap enabled applications can be coordinated to create a presentation—this includes applications that have not yet been created.
- **Fault Isolation:** A presentation may continue even if one of the data viewers or event receivers is down, although clearly that specific desired action will not take place.
- **Snooping:** Without the master presenter or any of the specific data viewers even being aware, the audience applet would allow users to follow the presentation from their laptops.
- **Adaptability:** PowerPoint was enabled as an Event Heap target by wrapping it with a simple Event Heap program that translated events to actions in PowerPoint.

While SmartPresenter was only recently completed, the ease of coupling applications and devices through the Event Heap has already allowed us to extend it. We have a set of wireless buttons in the iRoom that can be bound to any event, and we found that we were able to make a presenter control by simply binding the advance presentation event to one button, and reverse to another. Now presenters can walk around the iRoom as they

present, returning to the main web-based controls only if they need to jump to a specific point in the presentation.

4.3 The CIFE Suite

The CIFE group [15], who inspired our scenario from section 1.1, are a group of civil engineers working on construction management. They designed a set of viewers for their data that could be run on the various displays in the room, and have since built an interactive workspace of their own where they use the same applications:

- A construction site map that allows the selection of various view points in the construction site and then emits an appropriate view change event.
- A "4D" viewer that shows a 3D model of projected state of the construction site for any date during construction. It responds to events that change the view, select objects and zones (e.g. building 3), and change the date for the current model view.
- Another map viewer that highlights zones based on zone selection events.
- A web based viewer that displays tables of construction site information and emits zone and date selection events as table information is selected and listens for the same events to select information in the table.

All of the applications are essentially stand-alone, but communicate through the Event Heap. The 4D viewer was designed for use on a single-PC and was modified to use the Event Heap by adding around 100 lines of code. Since they use common event types, the various components of the suite retain their ability to coordinate while still being able to be brought up on any screen in the room. Since the components are loosely coupled, if no event source is running, or there is no event sink, it does not affect any of the application components currently in use. Much of the CIFE application is essentially plain HTML using the web event submission path; only the custom 4D viewer and the zone map viewer were coded specifically to communicate with the Event Heap, using the C++ and Java versions of the Event Heap respectively. To implement the top-down map mentioned in the scenario in section 1.1, the map and previews of the desired views were created in Macromedia Flash [17] with embedded URLs triggered by selecting a region of the map. This made it possible to create this new application with a minimum of development time.

Using the components they have constructed they have created a demonstration scenario that works almost exactly like the one presented in section 1.1.

5 Discussion

5.1 Experience With Robustness, Extensibility and Portability

Three of our goals for our coordination infrastructure were robustness, extensibility and portability. Since our code has been under development, and Java is still a rapidly evolving product, the system has not been as inherently robust as would be ideal. Still the system has been remarkably stable. While individual machines or the interactive workspace daemons on the machines have failed on many occasions, the rest of the infrastructure has continued to function correctly. During development the server itself hasn't always been stable. The automatic client-reconnect and quick web path to restart the Event Heap server and servlets have meant that it seldom has taken more than a few minutes to get the infrastructure back up and running after a problem. We have recently done some work on modular re-startability [6] that we hope to apply to the iRoom to make the system even more robust.

The Event Heap system has also worked out well for in terms of extensibility and adaptability to new platforms and legacy applications. The Python port took only a week or so for one graduate student to complete. The Event Heap servlets were similarly straightforward, although they took slightly longer to complete due to a lack of familiarity with Java servlets. We were able to create a wrapper for PowerPoint using Microsoft Office Automation that took less than a day once we figured out Office Automation. Now that we have standard template code for integrating OLE applications it should be easy to make most Windows applications valid interactive workspace components.

We are still in the preliminary stages of testing portability across other interactive workspaces. The CIFE group has been able to take the CIFE suite on the road with the aid of the *mbforward* tool for rerouting multibrowse events, and have recently deployed an iRoom of their own where they have been running the CIFE suite with minimal modifications. The Stanford Learning Lab (SLL) has done a preliminary deployment of the Event Heap and has the multibrowsing system up and running in their interactive workspace. Several other locations on Stanford campus are also testing use of the Event Heap and other software from our research group, including one actively used classroom. In early April, 2002, following an initial release in Summer 2001, we released our second set of installers for either setting up an interactive workspace with some of the basic programs we have developed, or to creating a development setup for users to write their own applications. The current installers were used for the latest installation in the iRoom

and in the other on-campus interactive workspace installations. The software has been downloaded for testing at locations around the world. The software is freely available for anyone to try at: http://iwork.stanford.edu.

The Event Heap code along with other infrastructure for interactive workspaces being developed by our group is also being released as Open Source to allow others to use and extend it for their own locations. The code is available through http://iros.sourceforge.net.

5.2 Limits and Performance of the Event Heap

Since the Event Heap was designed for the interactive workspace domain, it has some drawbacks that make it less useful outside of the space. Due to the use of a shared medium, it is difficult to scale the system to large numbers of simultaneously communicating entities. Among other things, this makes it unsuitable for Internet scale coordination across tens or hundreds of thousands of devices. While the indirect communication mechanism provides nice properties for our domain, it forces two hops of communication, which adds latency (although in [7] it has been shown that for a static communication pattern a properly implemented tuplespace will adapt over time to single-hop communication). Both latency and scalability in an interactive workspace are, however, bounded by social constraints and human factors which makes these drawbacks less of an issue. Specifically, latency on a local sub-net is small enough that even doubling it keeps the response fast enough to be perceptually instantaneous for humans. Similarly, scalability is bounded by the number of people that can meaningfully interact with one another and a set of devices to solve some problem.

The actual performance of the Event Heap has been satisfactory so far. A performance study with the server running on a Pentium II, 450 MHz with 256 MB of RAM shows that under the pathologically worst case scenario for our implementation, the system can provide sub-100 ms latency to 12 different applications each generating and receiving 10 events per second. In practice, the event generation rate due to user interaction with applications is likely to fall far below this. The quiescent latency of the system, including network time, is around 2 ms. Our implementation was done as a proof of concept, so we anticipate a more finely tuned system would be able to perform much better than this.

Finally, it should be noted that although generality was one goal of the Event Heap, this does not make it appropriate for all types of communication. Some data probably should not be sent through the Event Heap even if a suitably efficient implementation could be made. For example, streaming video through the Event Heap is inappropriate since the video stream does not represent a stream of events to which other applications could or should react. In general, the Event Heap should be used to communicate events where anonymity, failure isolation, and the ability to perform intermediation or snooping are useful capabilities. Thus broadcasting of state, or requesting that appropriate applications perform some action are appropriate uses of the coordination facilities provided by the Event Heap. For practical purposes, not all events of this nature can currently be routed through the Event Heap. In particular, due to latency concerns, we use an alternative communication infrastructure right now for routing mouse events in our pointer control system. Some initial tests with our most recent implementation have shown it to be capable of handling these events, so perhaps in the future even these will be routed through the Event Heap.

5.3 Open Research Issues

We plan to investigate the effects of flexible typing in this system, with the ultimate goal of producing a systematic framework for event intermediation to enable ad-hoc interactions. Clearly we also have a problem with collision on the EventType field between application writers that do not coordinate. For a single interactive workspace with a tight knit community this hasn't been a problem, but it needs to be addressed before our infrastructure can be more widely deployed.

We also plan to construct a set of flexible Event Heap managers to control application composition and coordination, both automatically and via human intervention, for arbitrary ensembles of Event Heap enabled applications.

Security and privacy are un-addressed problems, in part because we lack a social model to indicate what mechanisms would be appropriate in collaborative settings: there is a tradeoff between user convenience and authentication as is typical in security systems. To complicate matters, our legacy-OS building blocks have varying security and privacy models. Currently our security model is to firewall off the room, and keep it physically secured, while giving users in the iRoom, who are assumed to be trusted, full access. All communication among applications in the iRoom through the Event Heap is public.

We are starting to investigate tele-connection of interactive workspaces—in fact, the Stanford Learning Laboratory [23] is already starting experiments connecting their prototype interactive workspace to our own. We envision that each connected workspace will have its own separate Event Heap, with selective communication of certain events across heaps. We suspect that some sort of 'meta-Event Heap' might be a useful abstraction, with coordination between Event

Heaps in different interactive workspaces being analogous to coordination between processes running on machines.

6 Related Work

A large number of interesting and complex, yet non-interoperable, projects ([2][3][5][8][24]) are investigating room or work-area based ubiquitous computing. Each has uncovered important insights in ubiquitous computing but have yet to propagate and deploy their software significantly beyond the project's boundaries.

Two such project are the MIT Intelligent Room project and Microsoft Research's Easy Living project [5]. They are both looking primarily at how to incorporate intelligence into ubiquitous computing rooms. For example, networks of observers should be able to track where you are in a room, and do the appropriate thing based on voice commands and gestures. MIT's infrastructure framework is called *metaglue* [8] and is based on agents written in Java. Coordination between agents is done using RMI, but standard interfaces and automatic mechanisms for composing agents together are provided. The Easy Living project currently only provides ad hoc mechanisms for extending the capabilities of their environment. Neither project focuses on addressing dynamic heterogeneous environments, and our project is not attempting to build intelligence into the environment.

The i-Land project [24] at Darmstadt is investigating a physical environment that is almost identical to the one we have set up in the iRoom. They are focused more on design and human computer interaction concerns for room and building based ubiquitous computing. They use a Smalltalk based framework derived from COAST [20], which was originally designed for computer supported collaborative work (CSCW) among geographically distributed users each at their own computer. It is intended for applications to be written from scratch for this framework, and in fact the group at Darmstadt has done some initial work using the Event Heap in conjunction with their own system.

The Portolano project at the University of Washington is exploring how to enable working environments with computer infrastructure. Their current work is on an instrumented and enhanced biology lab workbench [3]. Their One.world [11] world infrastructure aims to enable pervasive computing in general, but is primarily an event driven system focused on easy check-pointing, aggregation and migration of applications.

Jini [25] provides a rendezvous mechanism for Java-based entities to begin coordinating with one another when they connect to a new network. It plus Java RMI could serve as a coordination model for an interactive workspace, but would have the drawbacks of RMI mentioned in section 2.4. A related technology distributed with Jini is JavaSpaces [21], which is another Java based tuplespace implementation. It would be possible to build the Event Heap on top of Javaspaces, but its Java-centric nature would make it difficult to provide clients on different platforms.

In [12], Hasha describes some of the requirements for a distributed object OS, mostly in his case for controlling homes filled with smart appliances, sensors and input/output devices. His proposal to use publish-subscribe meshes well with the function of the Event Heap, although, as discussed earlier, we believe tuplespaces to be a better suited starting point for an interactive workspace coordination infrastructure than publish-subscribe.

7 Conclusions

In this paper we have demonstrated that the Event Heap, a model derived from tuplespaces, is an appropriate general-purpose coordination system for interactive workspaces due to its portability, extensibility, flexibility, and ability to deal with heterogeneous environments. We identified several key extensions to the basic tuplespace model for this domain: flexible, typed, self-describing tuples, tuple sequencing, tuple expiration, support for default routing and query registration. To validate our choice we have implemented a version of the Event Heap. The system has been in use in our prototype environment, the iRoom, for over two years, and in the past six months has been deployed in several other locations. Application ensembles have been created and successfully run in the various workspaces. Our experience suggests that the loosely-coupled nature of our model makes it ideal for an interactive workspace, and we propose that it would also work well for many other ubiquitous computing situations.

Acknowledgments

The Interactive Workspaces project is the result of efforts by too many students to name, both in our research group and in collaborator groups from other departments. Susan Shepard and Brian Luehrs deserve special thanks for maintaining the iRoom and keeping it functional. See http://iwork.stanford.edu for an exhaustive list of participants and more complete project information. The work described here is supported by DoE grant B504665, by NSF Graduate Fellowships, and by donations of equipment and software from Intel Corp., InFocus, IBM Corp. and Microsoft Corp.

References

[1] Ahuja, S., Carriero, N., and Gelernter, D., Linda and Friends, *IEEE Computer*, August, 1986.

[2] Abowd, G., "Classroom 2000:An Experiment with the Instrmentation of a Living Educational Environment," IBM Systems J.,Vol.38,No.4,Oct.1999,pp.508-530.

[3] Arnstein, L. et al. Ubiquitous computing in the biology laboratory. *Journal of Laboratory Automatio*n, March 2001.

[4] Automation programmer's reference : using ActiveX technology to create programmable applications. Microsoft Press, Redmond, Wash., c1997.

[5] Brumitt, B., Meyers, B., Krumm, J., Kern, A. and Shafer, S.,. Easyliving: Technologies for intelligent environments. In *Handheld and Ubiquitous Computing 2000 (HUC2K*), September 2000.

[6] Candea, G. and Fox, A., Recursive Restartability: Turning the Reboot Sledgehammer Into a Scalpel, *In Proc. Eighth Intl Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Oberbayern, Germany, May 2001

[7] Carriero, N., Gelernter, D., Mattson, T., and Sherman, A., "The Linda alternative to message-passing systems", *Parallel Computing*, 20, 633-655, 1994.

[8] Coen, M., Phillips, B., Warshawsky, N., Weisman, L., Peters, S., and Finin, P. Meeting the Computational Needs of Intelligent Environments: The Metaglue System, *Managing Interactions in Smart Environments*,. Paddy Nixon, Gerard Lacey and Simon Dobson eds. Dublin, Ireland, 1999

[9] Davies, N., *et al*., *L2imbo: a distributed systems platform for mobile computing*. Mobile Networks and Applications, 1998. **3**(2): p. 143-56.

[10] Gelernter, D., and Carriero, N., Coordination Languages and their Significance, *Communications of the ACM*, Vol. 32, Number 2, February, 1992.

[11] Grimm, R., Anderson, T,. Bershad, B., and Wetherall, D., A system architecture for pervasive computing (PDF, 128 KB). In *Proceedings of the 9th ACM SIGOPS European Workshop*, pages 177-182, Kolding, Denmark, September 2000.

[12] Hasha, R., Needed: A common distributed object platform, IEEE Intelligent Systems. March/April 1999.

[13] Johanson, B., Ponnekanti, S., Sengupta, C., Fox, A., "Multibrowsing: Moving Web Content Across Multiple Displays," *Proceedings of Ubicomp 2001*, September 30-October 2, 2001.

[14] Leler, W., *Linda meets Unix*. Computer, 1990. **23**(2): p. 43-54.

[15] Liston, K., Kunz, J., and Fischer, M., "Requirements and Benefits of Interactive Information Workspaces in Construction," The 8th International Conference on Computing in Civil and Building Engineering, Stanford, USA, 2000.

[16] Murphy, A.L., G.P. Picco, and G.C. Roman. LIME: a middleware for physical and logical mobility. in CF- 21st International Conference on Distributed Computing Systems. 2001. Mesa, AZ, USA: Los Alamitos, CA, USA : IEEE Comput. Soc, 2001.

[17] Macromedia Corporation, Macromedia Flash, http://www.macromedia.com.

[18] ProxiNet Inc. ProxiWeb browser. See http://www.proxinet.com.

[19] Rowstron, A.I.T. and A.M. Wood. *BONITA: a set of tuple space primitives for distributed coordination.* in *Thirtieth Hawaii International Conference on System Sciences*. 1997. Wailea HI USA: Los Alamitos, CA, USA : IEEE Comput. Soc. Press, 1997.

[20] Schuckmann, C., Kirchner, L., Schummer, J., and Haake, J.,. Designing object-oriented synchronous groupware with COAST, *ACM Computer Supported Collaborative Work,* November 1996.

[21] Sun Microsystems Labs, JavaSpaces Specification, http://www.sun.com/jini/specs/js.pdf.

[22] Smart Technologies SMART Board, http://www.smarttech.com/smartboard/.

[23] The Stanford Learning Laboratory, http://sll.stanford.edu/.

[24] Streitz, N.A. et al., i-LAND: An interactive Landscape for Creativity and Innovation. In Proc. ACM Conference on Human Factors in Computing Systems (CHI '99) , Pittsburgh, Pennsylvania, U.S.A., May 15-20, 1999. ACM Press, New York, 1999, pp. 120-127.

[25] Waldo, Jim, Jini Technology Architectural Overview, Sun White Paper, 1999

[26] Weiser, M., The computer for the twenty-first century. *Scientific Americ*an, pages 94–100, September 1991.

[27] Wyckoff, P., McLaughry, S. W., Lehman, T. J. and Ford, D. A., TSpaces. IBM Systems Journal 37(3). Also available at http://www.almaden.ibm.com/cs/TSpaces.