# 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 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 that a tuplespace model with several extensions is ideal for coordination in this environment. We present a prototype implementation of such a model called the Event Heap. Finally, we show that the system has performed well in actual use over the last year and a half 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 [23]. 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 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 coordinating these devices. Based on our experience with a prototype room-based ubiquitous computing environment (interactive workspace) that we have deployed, the realities of these environments make the existing models incomplete or inadequate. Interactive workspaces systems must be able to tolerate a dynamic environment, as portable 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 room-based ubiquitous computing environments that are technology-rich and consist of interconnected large and small displays and multi-modal I/O devices, where people gather to do naturally collaborative activities such as design reviews, brainstorming, etc.. 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 identified the following desirable features:

- Applicable to many different types of ubiquitous computing applications.
- Portable across installations, provided some infrastructure for determining workspace-specific information is also provided.
- Friendly to existing languages and environments, and portable to new ones, making 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:

First, we identify the need for a general-purpose coordination system, in the spirit of [9], in an interactive workspace. Further, we propose that a tuplespace system is well suited for this role. We will show why a desire to integrate legacy devices in an interactive workspace and other "engineering" constraints lead to a tuplespace coordination model.

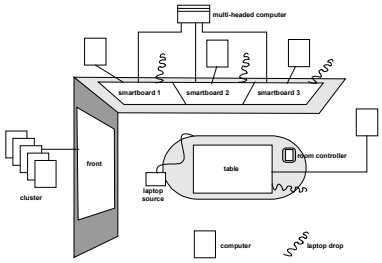
Second, we explain the extensions to the basic tuplespace model that we have found necessary in an interactive workspace, and why they are needed. Specifically, the extensions are self-describing tuples, flexible typing, typed tuples, tuple sequencing and tuple expiration.

Finally, to we present the Event Heap, an implemented and deployed tuplespace based coordination substrate for a real interactive workspace, and discuss our experience with applications using the infrastructure across a wide variety of devices and software platforms over the past year and a half. The abstraction and API has proven to be well suited for this domain, and the current implementation has been able to provide for all but low-latency direct feedback communication..

### 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 SmartBoard [19] touch screens 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.




(a) iRoom Layout

(b) Construction Management in the iRoom

**Fig. 1**. The iRoom

We now provide a scenario set in the iRoom that reflects how research groups collaborating with us hope to use interactive workspaces. 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 [12], 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 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 are not specifically 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 presented in this scenario.

The rest of the paper proceeds as follows. We begin by clarifying *interactive workspace*, and the types of interactions we have observed to be common in them. We then discuss the model of coordination languages, and our choice of the tuplespace model. In order we next present the Event Heap itself, and then applications implemented using the Event Heap. We conclude with some future research directions 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 determining which coordination model best facilitates this composition, such that the user has the impression of using one distributed application.

### 2.1 Coordination-Based Programming Background

In [9], Gelernter and Carriero proposed that computation languages and coordination languages should be thought of as orthogonal. Computation languages express how calculations proceed, and coordination languages express the interaction between autonomous processes. By choosing a computation language, say C++, and a coordination language, say a set of message passing primitives, you can entirely express an autonomous process and its interaction with other processes. They further argue that general-purpose coordination languages which can express any type of coordination are superior to task specific languages that may be useful, for example, only for fine-grained parallel computing.

Gelernter and Carriero position the Linda [1] tuplespace model as an example of such a general-purpose coordination language. The tuplespace model consists of just three key operations, out(), in(), read() and one data type, a tuple. Tuples are a set of ordered typed fields, where each field may either contain a value, or be undefined. The primitive 'out' puts a tuple into an abstract space, the tuplespace, which is visible to all processes. The 'in' and 'read' processes remove or copy a tuple, respectively, from the space that matches a template tuple where explicit values are used for some fields, and wild cards for others.

Gelernter and Carriero 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 like Linda, 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.

## 2.2 Tuplespace Advantages for Interactive Workspaces

These same features of tuplespaces, portability, heterogeneity, economy and flexibility, are the same as ones we identified as being important for an interactive workspaces coordination model. This makes the tuplespace model a good starting point for such a model. Simplicity, ease of coordination, and good robustness and failure isolation properties are other important features for an interactive workspace coordination model, and characteristics of the tuplespace model.

First, tuplespaces are simple and flexible, and are therefore easy to deploy on many devices and platforms. Since there are only three primitives, it is fairly simple to port the interface to new platforms. This also makes it easy to add support for tuplespaces into the code base for existing applications, or to make wrappers to programmatic interfaces for applications that have no source code. With tuplespaces, coordination state is stored in the infrastructure instead of in individual clients. This property is another one that makes client code small and straightforward to implement for new platforms and devices, even relatively impoverished ones. Since tuplespaces are general-purpose, other coordination types can be implemented on top of it if they are more appropriate for some task. For example RPC can be implemented as two tuplespaces calls on the calling process, and two on the callee. Finally, while marshalling and un-marshalling are required, the tuplespace code running on the client need only implement marshaling into and un-marshaling from the basic tuple format.

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. The rendezvous mechanics for applications are also straightforward, and are aided by the following three key 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 they 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 [14]. 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.

Failure isolation in tuplespaces is naturally achieved since receivers and senders don't directly interact. 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. Applications can therefore retrieve communications even if they were transmitted while they were down. This

persistence also makes it more difficult to create application ensembles that are dependent on start up order since all tuples will remain in the tuplespace until the appropriate party starts and picks it up.

Tuplespaces do, however, have some well known drawbacks. Scaling is an issue since all participants communicate through a shared medium. For the interactive workspace domain this is not a big problem since workspaces will only contain on the order of 10s of users and 100s of individual processes. The decoupled nature of the tuplespace means that all communication takes two hops. Our intent to provide coordination between applications over human-scale latencies and the speed of current computers and networks combined have meant that this is not a problem for us. Also, in [7] it has been shown that a properly implemented tuplespace will adapt over time to have the same number of hops (one) over time as a message-passing system, and a similar analogy could be made for RMI/RPC, so we don't believe the latter is a fundamental problem with the tuplespace model.

### 2.3 Adapting Tuplespaces for Interactive Workspaces

While the tuple space model is a good general-purpose system for coordination, we found that certain extensions were necessary for the interactive workspaces domain:

**Self-describing Tuples:** Since ensemble components aren't necessarily designed to work together, it is important to have tuples be self-describing so users can figure out the intent of tuples by browsing through the tuplespace. This specifically means that every field has a name in addition to the standard type, and data.

**Flexible Typing:** In the standard model, the number of fields in a tuple and the order of fields is significant. For flexibly typed tuples, both field order and event size are ignored and matching is done by name instead. This allows applications to create extensions to standard event types that include extra information without breaking older applications. This is important since the collection of hardware and software in use is continually evolving. We have used it in the iRoom to add flags to tuples to support devices with new capabilities without breaking compatibility with deployed applications.

**Typed Tuples:** In part as a side effect of using flexible typing, and in part because application writers won't necessarily coordinate tuple structure, tuple fields with the same name will not always have the same semantic meaning. For example, 'xPos' may be a screen offset in one application and a position in a 3D model for another. This problem can be avoided by including in all tuples a 'TupleType' integer field which implies the presence of certain fields and the semantics of those fields. This provides a useful compromise between strong and weak typing. A problem still exists when application writers choose the same 'TupleType' value for tuples with dissimilar semantics, but having type greatly reduces the problem of name collision.

**Tuple Sequencing:** Traditionally, if multiple tuples exist that match the template tuple on a 'read' or 'in' operation, any of the matching tuples can be returned. Tuple sequencing means that receivers always get the earliest matching tuple they haven't seen yet. 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.

**Expiration of Tuples:** Since sources and receivers are decoupled, a source need not have a receiver to continue running. This may cause tuples to build up in the tuplespace, and for a real world system this means that the performance steadily decreases. To ameliorate this problem, all tuples should be 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 tuples long after the triggering occurrence. Thus, tuples should expire after a human would no longer expect the causal action to take place. 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 but users have forgotten about the request is not acceptable.

For an interactive workspace, we want users to be able to dynamically compose the application components they are using into an ensemble, which differs from the original intended use of tuplespaces: 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 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 our main contribution 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. One difference is that events in publish-subscribe systems have no persistence, so there is no inherent way for restarting applications to pick up recent events. This makes it more difficult to keep things running through a failure. In some publish-subscribe systems, receivers also need to track new senders to decide whether or not to subscribe, while for a tuplespace blocking on a match will get you the appropriate tuple, be it from an old or new source. Finally, for efficiency most publish-subscribe systems only broadcast events when there are receivers. This makes it more difficult to snoop on what sorts of tuples are currently, or have recently been sent.

Both RMI/RPC and message passing suffer from a similar set of drawbacks in the interactive workspaces domain when compared to each other. 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. For both RMI/RPC and message passing, either the method interface or message format, respectively, needs to be agreed upon ahead of time by all parties, and working around this requires explicit adaptor processes. Since communication is direct, getting programs not designed to work with each other to rendezvous is more difficult. In particular snooping and intermediation are not very well facilitated by the basic coordination model. Finally, for both of these models it is easier for ensemble

writers to create start up order dependencies if calling processes or a message-generators are not programmed to retry if the target process is not yet started to receive the call or message.

# 3 The Event Heap Implementation

The Event Heap is our implementation of a tuplespace based coordination system for interactive workspaces. It is built on top of TSpaces from IBM [24], a Java based tuplespace system that includes many extensions to the basic tuplespace model. One feature of TSpaces tuples is that they can be self-describing—one of the key extensions we identified for using a tuplespace in an interactive workspace in section 2.3. The TSpaces system is client-server based with the actual state of the tuplespace stored on the server machine. While the TSpaces server is a single point of failure, individual Event Heap client applications automatically reconnect if the server goes down and is restarted. This combined with a dedicated web server that handles requests to restart the TSpaces server and other Event Heap server applications minimizes the problems with server failure.

In the Event Heap, therefore, 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, and emphasizes that, as described in section 2.3, the semantics are somewhat different than that of classic tuples. In the remainder of the paper 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.

Due to the relatively high latencies (100 ms to 1 s) we see from the TSpaces system, the current version of the Event Heap is used primarily for coarse high-latency coordination between applications running in the iRoom. Nonetheless, there is a need for coordination within the workspace on a finer scale, for example to route mouse/pointer events between different devices. We have written a system for fast routing of events in these cases and in fact use it for a flexible mouse and pointer re-routing system we use in the iRoom. We contend that the Event Heap could be combined with this fast path to create a single system that would be able to handle both the current coarse-coordination, and the lower latency domains.

## 3.1 Event Format Description

The basic event used by the Event Heap is a TSpaces 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.

**Table 1.** Standard Event Heap Fields

| Field Name | Field Type | Meaning |
|---|---|---|
| EventType | Mandatory | A string that uniquely identifies an intended event type, and is associated with the declaration of extra fields associated with this type. |
| SourceID | Mandatory | The unique identifier of the sender of this event. |
| TimeToLive | Mandatory | Milliseconds after submission when the event will be removed. |
| EventName | Optional | A plain-english description of this event type. |
| PersonID | Optional | An identifier for the human who generated or is associated with this event. |
| GroupID | Optional | The application group for which this event is intended. |
| TargetID | Optional | The ID of the desired target of this event. May or may not be the SourceID of the target. |
| EventHeap Version | Internal | The version type of this event. Used to differentiate fields used and semantics between versions. |
| SessionID | Internal | Used for sequencing events. |
| SequenceNum | Internal | Used for sequencing events. |

In addition to EventType (previously referred to as 'TupleType') and TimeToLive which were mentioned in section 2.3, there are several other standard fields. These fall into the categories of mandatory, internally used and optional. Table 1 lists these fields, and briefly describes their use.

### 3.2 Event Retrieval

The Event Heap provides additional operations to retrieve events beyond the basic operations of 'out' and 'read.' 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. Finally, there is a 'snoopEvents' call which retrieves events without effecting sequencing.

**Table 2.** Event Heap Communication Types

| Comm. Type | Effect | Fields to Set |
|---|---|---|
| Dedicated-Receiver | Receives from specific source(s) | SourceID of receiver |
| Dedicated-Source | Sends to specific receiver(s) | TargetID of source |
| Dedicated-Link | Send between specific source and receiver | SourceID of receiver and TargetID of source |
| Constrained to Group | Events only seen within app group | GroupID of all apps in group |
| Restricted by Person | Receive only events created by one person | PersonID of receiver |

Control over which applications receive certain events can be accomplished by having event sources set the SourceID, TargetID, GroupID and PersonID to specific values, and having clients match on specific values for those fields. For example, a client can receive from a specific source by setting the SourceID field in their template event to a value corresponding to the source from which they desire to receive the event. A variety of communication modes are possible depending on how the fields are set, as shown in Table 2.

### 3.3 Event Sequencing

To perform sequencing, each source tags every generated event with a SourceID, a SessionID, and a SequenceNum (sequence number). The SourceID needs to be unique among all sources participating in the Event Heap, and may be either specified by the application, or assigned automatically.[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.

On the receiver side, sequencing is accomplished using the query syntax of TSpaces, which allows database-like queries to find matching tuples. For every source and event type, receivers keep track of the session ID and sequence number of the most recently retrieved event. In addition to querying to match the application specified field values, when an event is retrieved the client code queries for events of the given type that are either from new sources, from current sources with a new session (which have presumably restarted), or are from a current source and session, but have a higher sequence number than the last seen event of this type from the given source. This combined with TSpaces FIFO option insures that applications will always receive the earliest event that is newer than the last event of the given type that was retrieved.

[1] Automatic assignment is done by choosing a random integer value between 1 and 2 billion, which is assumed to be virtually unique.

### 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 3:

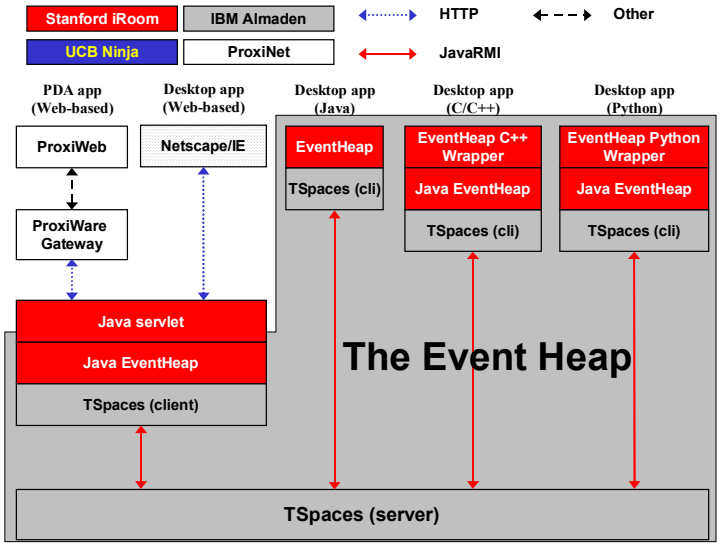



**Fig. 2.** Methods of Accessing the Event Heap

The main implementation of the Event Heap is in Java, which is also the native language for TSpaces. Other pathways are implemented as wrappers of the Java version in the native language, thus there is only one code base where actual client logic is maintained. The total size of the Event Heap extension set to TSpaces is about 15KB, so it is deployable to most devices. 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. In particular this path has proven useful in allowing PDAs to participate in controlling the iRoom, since even Palm-type devices have web browsers available today (for example the ProxiWeb [15] 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.

## 4 The Event Heap In Practice

In this section we present some applications built on the Event Heap that we use in the iRoom and relate some of our experiences with the robustness, extensibility, and portability of the Event Heap over the year and a half or so it has been in service.

### 4.1 Applications

Ten to twenty applications have been written which use they 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.

**Multibrowsing**
Multibrowsing 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 its TargetID, 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. Executable applications can also be submitted, in which case they are run by the multibrowse daemon.[2]

Using the web path to the Event Heap, users are able to encode requests as links on web pages to pull up other web pages, data, or applications on the other machines in the iRoom. 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. 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 set of TargetIDs and automatically routes them to different machines. Using this mechanism the CIFE group [12] has been able to demonstrate multibrowse scenarios tailored for the iRoom on a set of laptops for demonstrations in other locations.

**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,

[2] This is a security hole, but we minimize the risk by fire-walling the iRoom sub-net.

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.

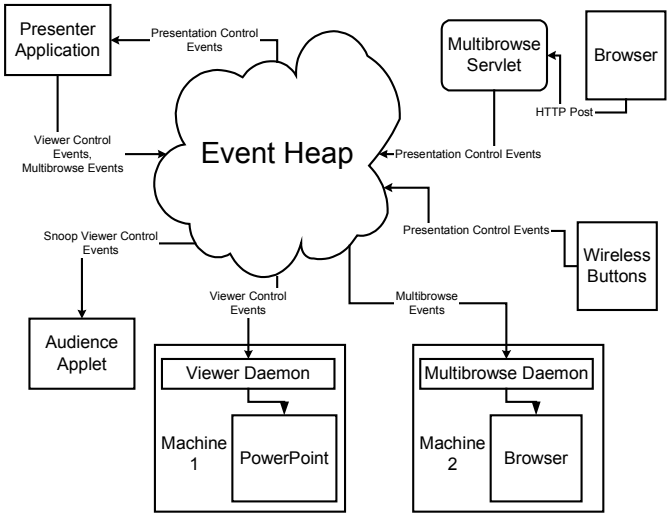



**Fig. 3.** Application Paths into the Event Heap

We can also 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 4 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 will 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.

**The CIFE Suite**

The CIFE group [12], 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:

- 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 [13] 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.

### 4.2 Experience With Robustness, Extensibility and Portability

Three of our goals for our coordination infrastructure were robustness, extensibility and portability. Since both TSpaces and Java are still rapidly evolving products, they are not 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. TSpaces can become unstable under high tuple loads and after certain transaction sequences. The automatic client-reconnect and quick web path to restart the TSpaces server and Event Heap 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 us 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.

So far we have the least experience with portability since there are not many interactive workspaces in existence. 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. 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. We have just completed and successfully tested a set of three Windows installers, one for developers, one for interactive workspace client machines, and one for an interactive workspace server machine. The latter two can be used to set up an interactive workspace with multibrowsing and projector control, and the former allows a programmer to create an 'interactive workspace on a machine' to use for development. We plan on deploying this to select groups inside and outside Stanford in the coming months, and eventually plan on providing the system as Open Source for anybody to download and use.

# 5 Lessons and Suggestions For Future Work

## 5.1 Alternative Implementations

TSpaces provides a powerful tuplespace framework, but has a relatively high latency, particularly when using querying (which we use for sequencing). It has been suggested to us [17] that using a high-performance, off-the-shelf commercial database would likely solve the performance problem, and give us persistence for free. We are investigating this option, but we note that fast restart is an important property for high availability in our system; most moderately-priced commercial systems do not necessarily afford this property. We are also considering adding event streams to the Event Heap API, and using our event-fast path system under the covers to insure these streams have low latency.

### 5.2 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 as described in [14]. 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 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 is an unaddressed problem, in part because we lack a social model to indicate what security 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 differing security 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.

We are starting to investigate tele-connection of interactive workspaces—in fact, the Stanford Learning Laboratory [20] 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][21]) are investigating room or work-area based ubiquitous computing. Each has uncovered important insights in ubiquitous computing but have yet to propagate and deploy them 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 [21] 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 called COAST [16], which was originally designed for computer supported collaborative work (CSCW) among geographically distributed users each at their own computer. As far as we know applications must be written from scratch using this framework in order to run in their room, and applications must be designed to coordinate with one another.

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 [10] world infrastructure aims to enable pervasive computing in general, and it may be possible to build a future version of the Event Heap on top of their system.

Jini [22] 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 is JavaSpaces [18], which is similar to TSpaces, but with simpler semantics. Like TSpaces, JavaSpaces implements a tuplespace in the Java environment. We chose to use TSpaces since we needed its querying semantics for sequencing, but we believe the Event Heap could also be easily built on top of the JavaSpaces system.

In [11], 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 we believe the temporal persistence of tuplespaces make them slightly more useful than publish-subscribe.

# 7 Conclusions

Ubiquitous computing is fast becoming a reality as portable and embedded hardware along with wireless networking become more common. Unfortunately the software platforms for coordinating interaction across devices are not as mature. We looked specifically at an interactive workspace, which is a room or similar environment with embedded computational capabilities, and a heterogeneous collection of applications and devices. We chose the tuplespace model as 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: self-describing tuples, flexible typing, typed tuples, tuple sequencing, and tuple expiration. To validate our choice we have implemented the Event Heap, a system built on IBM TSpaces that adds the aforementioned extensions. The system has been in use in our prototype environment, the iRoom, for over a year and a half, and many application ensembles have been created and successfully run in the space. Our experience suggests that the loosely-coupled nature of a tuplespace model makes it ideal for an

interactive workspace, and we propose that it would 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 deserves special thanks for maintaining the iRoom and keeping it functional. See http://graphics.stanford.edu/projects/iwork 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] G.Abowd,"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] Larry Arnstein 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] George Candea and Armando Fox, 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] Gelernter, D., and Carriero, N., Coordination Languages and their Significance, *Communications of the ACM*, Vol. 32, Number 2, February, 1992.

[10] Robert Grimm, Tom Anderson, Brian Bershad, and David Wetherall. 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.

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

[12] 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.

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

[14] Michelle Munson and Armando Fox, "Dynamic Control in Tuple Spaces for Sustainable Evolution in Pervasive Computing Applications". Unpublished abstract.

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

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

[17] Shafer, .S., Personal communication. 2001.

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

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

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

[21] N.A. Streitz 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.

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

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

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