The Event Heap: An Enabling Infrastructure for Interactive Workspaces

Brad Johanson, Armando Fox, Pat Hanrahan, Terry Winograd
Stanford University
Gates 3B-376
Serra Mall
Stanford, CA 94305-9035
{bjohanso,fox,hanrahan,winograd}@graphics.stanford.edu

Abstract

As computers and large displays become cheaper, additional modes of human computer interaction are becoming possible. One can now set up an *interactive workspace* in which multiple computer displays and input devices are simultaneously visible and usable by one or more users. Unfortunately, the well-known event queue metaphor, which works well for a single user sitting in front of a single computer using a GUI, breaks down in such an interactive workspace. We propose the Event Heap as a novel mechanism by which multiple users, machines and applications can all simultaneously interact as consumers and generators of system events. The Event Heap is flexible, robust to failure of individual interactors, and sufficiently lightweight to be integrated easily with "legacy" single-user-GUI applications.

Keywords: Multidevice interaction, event stream, tuple space, interactive workspace

1 INTRODUCTION

"[A]n object-network OS…aimed at the intelligent-environment R&D problem space…would enable researchers to focus more on investigating intelligent environments and less on developing the infrastructure to support those investigations."

---*R. Hasha,* Needed: A common distributed-object platform, *IEEE Intelligent Systems, March/April 1999*

Today's standard environment for computer interaction consists of a keyboard, mouse and monitor. This works well for individual work, but it is not sufficient when multiple people interact simultaneously, or when multiple devices of different capacities need to be used together. An alternative configuration is an *interactive workspace*: a room or other work area with large projected screens that can be seen and controlled by all participants. Users interact with the displays using touch sensitive surfaces, wireless mice, tablet input devices, or sophisticated tracking techniques. In addition, laptops, PDA's and other portable computing devices with wireless networking capabilities may be brought into the room and used, as described in [4].

The traditional model for interaction in single-machine, single-user systems today is the event queue. In this model, all system events, including input events such as mouse and keyboard activity and application-level events such as window redraws, are placed in a queue and centrally dispatched to the appropriate foreground window. This system begins to break in a multi-person multi-device workspace, in which events are being generated by multiple entities and dispatched to different applications. The decentralized nature of an interactive workspace also means that the set of active event consumers and producers may be frequently changing, for example, as users with mobile computers enter and leave the environment while applications continue to run.

The Event Heap is a software infrastructure designed to provide for interactive workspaces what the event queue provides for traditional single-user GUI's. The system is an extension of TSpaces, a tuplespace system from IBM Research [22]. It is bulletin-board based, and applications may post and retrieve events of interest.

In addition to supporting interactions among multiple users, machines, and applications, the Event Heap was expressly designed to function well in the decentralized environment of an interactive workspace. Because these workspaces comprise a heterogeneous collection of machines and must support existing ("legacy") applications as well as purpose-written applications, we made the Event Heap fairly simple and portable across platforms. Because the large number of independent software and hardware entities in an interactive workspace make for a potentially fragile distributed system, we designed the Event to isolate failures: A misbehaving application using the Event Heap will not crash other applications or the interactive workspace as a whole, although clearly the functionality provided by the crashed application is lost.

The Event Heap is a working system and has been in use for over six months in an experimental interactive workspace called the Interactive Room. Our experiences so far suggest that the Event Heap successfully provides the functionality just described.

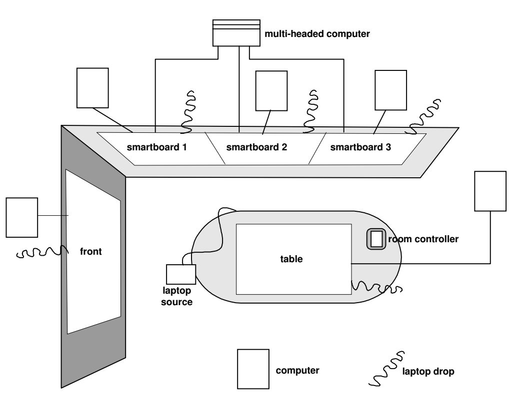
In the rest of the paper we present the Event Heap system in more detail and describe our experience using it as the primary infrastructure for Stanford's Interactive Room, or iRoom, a prototype interactive workspace. In Section 2 we start by giving an example of using an interactive workspace to further motivate the need for the Event Heap. In Section 3 we list the design goals for the Event Heap. Section 4 gives the specifications for the Event Heap and describes our implementation. In Sections 5 and 6 we describe various substantial applications in production use that exploit the Event Heap, and the lessons we have learned so far from implementing and using them on a regular basis. We describe related work in Section 7, outline our continuing research agenda in Section 8, and evaluate our contribution in Section 9.

2 MOTIVATING EXAMPLE

We set our scenario in the Interactive Room, or iRoom, which we have built to investigate interactive workspaces. As shown in the schematic of Figure 1, the iRoom features three rear projected SmartBoard [16] touch screens along one wall, a bottom projected table, and a front screen. The three SmartBoards may be driven by individual machines or by a single multi-head machine. Each display may also be driven from a laptop drop cable exposed at the table. All of the machines driving displays are Windows machines in order to allow legacy applications to be run in the room. In addition, the room has wireless LAN coverage, which allows laptops or PDA's to communicate with the other machines in the room.

Figure 1 - Layout of the iRoom

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 [8], but



similar scenarios apply for many domains requiring multi-person collaborations and interacting with large amounts of data.) Some project information is stored on laptops and PDA's they bring to the iRoom; other information is already stored on the network or on the computers in the room. Upon entering the room they use a touch sensitive tablet to turn on the projectors for the three touch screens in the side wall. The project manager has prepared a meeting outline earlier, which he calls up by using the touch screen on the left-most display. The outline lists the various topics, of the meeting, and each topic is a link that calls up appropriate information on the various screens in the room. Figure 2 shows a photograph of the iRoom in use.

Figure 2 – Construction Management in the iRoom

The first few topics don't require more information, but the fourth topic is a change in the schedule for one aspect of the construction. When the project manager taps the link for that topic, the table display turns on automatically and shows an aerial view of the building site. At the same time, a 3-d model of the site at the time of the schedule change is brought up on the middle touch screen.

As they discuss the scheduling change, they use a wireless pointer to select points on the map on the table display. Each time they do so, they see new views of the site in the 3-d viewer to visualize how the re-ordering of construction will affect the site construction progress. During the discussion they bring up supplemental data from their laptops and display it on the screens in the room. They are also able to redirect their laptops' pointers and keyboards to control the machines driving the displays when they need to directly control applications from where they are sitting.

3 DESIGN GOALS

From our example we can extract certain capabilities that would be desirable in an underlying infrastructure to support similar interaction scenarios. One is the ability to control the displays and environment through different paths including directly through a touch tablet, or indirectly by having the screen turn on when there is data being displayed on it. Users need to be able to display data on any of the screens in the room based on their current topic of discussion. Applications need to be able to communicate back and forth to coordinate among each other. Users need to be able to control displays directly using touch, or indirectly using remote keyboards, pointers and portable computers. These observations led to our design goals for the Event Heap:

- **Multi-User, Multi-Application, Multi-Machine:** Multiple users must be able to interact at the same time, either with different applications, or within the same application, regardless of which machines are running the applications.

- **Heterogeneous Machines, Legacy Support:** As much as possible, the infrastructure should allow the deployment of interactive workspaces using a variety of machines and operating systems. This allows groups to use legacy (single-user) applications in the interactive workspace despite the fact these were designed for a single-user environment. We also want users with PDA's and other personal computing devices to be able to participate in the workspace using their devices.

- **Failure Isolation:** The failure of any given application should not effect any applications with which it is not interacting, and should only effect those with which it is interacting to the extent that its functionality is lost. This allows the entry and exit of transient machines, as well as protecting the system as a whole from failure in any given member.

In the next section we present the Event Heap, a system designed to satisfy these goals.

4 THE EVENT HEAP ARCHITECTURE

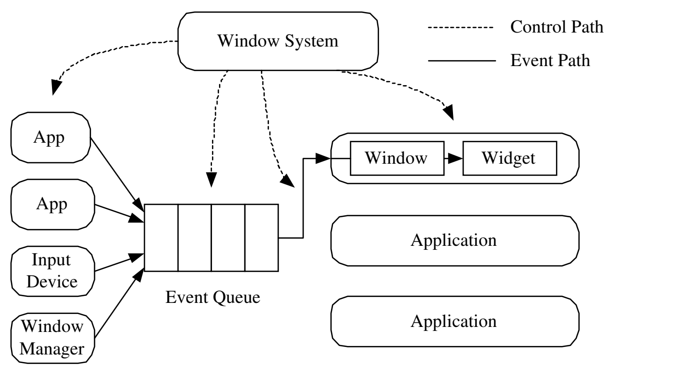
The Event Heap is a loosely typed, bulletin-board based message exchange system. Applications can post events of a certain type or request events that match certain criteria. Applications need not know about one another ahead of time to exchange events—they merely need to be producer and consumer of matching types. Since there is no direct communication between producers and consumers of events, the system as a whole does not rely on the presence or robustness of specific interactors: Events that are not consumed time out and are automatically erased from the Event Heap without further effect. The system is also inherently non-blocking: any number of events of any type may be posted and retrieved concurrently, and independent collections of applications can interact without cross-interference.

The Event Heap is not intended to replace the event queues of the individual machines in an interactive workspace. The level of control we desire is inter-application synchronization, applications controlling other remote or local applications, and redirection of input devices among the machines in workspace.

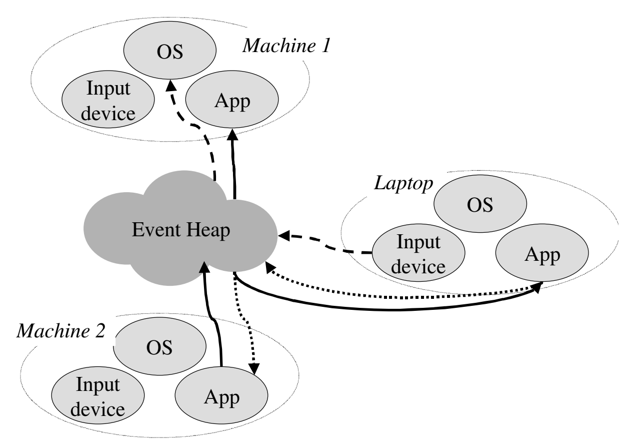
Since most interactive workspaces are likely to be composed of machines running traditional operating systems and legacy applications, the Event Heap API makes it easy to integrate them seamlessly. The interface provided by the Event Heap is simple enough to allow for easy integration into legacy applications for which we have the source code. It is also relatively straightforward to use application hooks (such as Visual Basic or VBScript) to allow legacy applications to participate in the interactive workspace by using the Event Heap. We have also provided a proxy-based connection that allow links on web pages to submit events into the system, allowing the construction of simple web interfaces that interact with the entire workspace.

4.1 Event Queue versus Event Heap

Before going into the details of the Event Heap, it is important to understand concretely why an event queue is not appropriate to accomplish the goals set out in the previous section. Figure 3 shows schematically the difference between an Event Queue and the Event Heap.



(a) Typical Event Queue for a Windowing System



(b) The Event Heap

Figure 3 - Event Queue versus Event Heap

A typical event queue that might be used for a workstation windowing system is shown in Figure 3a. In such a system applications, input devices and the window manager submit events into the queue. When an event reaches the front of the queue, it is read by whichever application is currently the target of the queue (usually the application that is in focus). The windowing or operating system explicitly manages the connection of event generators to the queue, the current destination of events in the front of the queue, and the chain of event recipients. The monolithic nature of the queue makes it difficult for the queue to handle multiple distinct users, since only one application is the target of events at any given time.

The Event Heap is shown in Figure 3b. In this case there is no explicit serialized flow of events through the system. An application on machine 2 can post events to be consumed by other applications on both machine 1 and the laptop. At the same time, the mouse on the laptop can be controlling the display of machine 1 by posting a series of mouse events to be consumed by the operating system on machine 1. Finally, there may be a direct exchange of events between the applications on the laptop and machine 1. The inherently many-to-many connectivity of the Event Heap allows it to function better in the distributed multi-machine, multi-application, multi-user environment of an interactive workspace.

4.2 TSpaces and Tuples

The Event Heap is built on top of the TSpaces (Tuple Spaces) system from IBM Research [22]. TSpaces is an implementation of tuple spaces as first described by Cisnero and Gelerntner for the Linda system [1]. Many of the concepts used in the Event Heap are derived from the underlying TSpace, so we briefly describe that system here.

A tuple is an ordered collection of fields. Each field has a type, name and value, any of which may be left undefined. Because the fields are ordered, each field has a unique index. Because the name, type and value are stored with every instance of the tuple, the tuple is self-describing—it is not necessary for the consumer of a tuple to know in advance how to “unmarshal” it.

A tuple space is a conceptual space into which tuples can be placed and from which tuples can be extracted by an application. To submit something to a tuple space, the source application fills in fields with the desired names, types and values, then calls an appropriate routine to deposit the tuple in the tuple space. Applications that are interested in certain tuples retrieve them from the tuple space by submitting a template tuple to the tuple space. The tuple space then returns all tuples that have the same type, name and value for fields that were explicitly defined in the template; fields that were left undefined in the template are filled in with types, names and values based on the matching tuples found in the tuple space. Tuple extraction may be destructive (the matching tuple is removed from the tuple space) or nondestructive (a copy of the matching tuple is returned).

While a tuple space may be run on one machine, it is most powerful when used by clients on multiple machines. In this case the tuple space becomes a clearinghouse in which applications running on different machines may exchange data with each other.

4.3 Event Format Description

The basic event used by the Event Heap is similar to a TSpace tuple, and in our implementation we use tuples directly to carry events. As with the tuple, an event is characterized by fields that contain names, types and values. Unlike tuples, we treat the fields as an unordered collection, and the Event Heap API’s allow references to tuple fields only by name, not by index. In addition, template events need only specify some combination of name, type and value for fields that are required in returned events to get all events that have those fields. This differs from the basic TSpaces semantics, which also requires specifying the correct field ordering, field types and total number of fields in the returned event. Our actual implementation uses various TSpaces query functions, which allow more flexible methods of tuple retrieval, to implement this functionality.

Event types should be flexibly defined, so that extending the event structure, either as part of the evolution of an event type or as a dynamic extension by a specific application, does not break existing consumers that do not understand the supplemental information. This is the main reason for our loose definition of an event as an unordered collection of fields. Since matches are done on only those fields desired by the recipient, additional fields that may be in use as extensions in a particular event will not prevent its delivery. For example, all mouse events may have x, y and button state as required fields. If a mouse with scroll wheel is added to the system, the generator of events for it could add another field with scroll wheel position without preventing the delivery of the basic information to receivers that only understand the old format. Listeners aware of this field, however, could check for its presence in any retrieved event and use the values found therein. A systematic treatment of the benefits of flexible typing for evolvable systems appears in [11].

Every valid Event Heap event has some mandatory fields, with at least name and type defined.. Some of the fields (marked ‘*’ in Table 1) must also have their value set. Table 1 lists the mandatory fields, and briefly describes their use:

Field Name	Meaning
EventType*	A string that uniquely identifies an intended event type, and is associated with the declaration of extra fields associated with this type.
SourceID*	The unique identifier of the sender of this event.
TargetID	The ID of the desired target of this event. May or may not be the SourceID of the target.
PersonID	An identifier for the human who generated or is associated with this event.
GroupID	The application group for which this event is intended.
SequenceID	Used for ordering events.
NumAccesses*	Number of times this event may be read before it is deleted.
TimeToLive*	Milliseconds after submission when the event will be removed.
TimeStamp*	Time when this event was last submitted to the Event Heap

Table 1 - Required Event Heap Fields

The required fields fall into several different categories. The first and most important field is the EventType, which must always be set by the source. Each EventType maps to a specific set of additional fields that will be in this event. SourceID is used to identify the sender of an event, and can be used by clients to receive events from only one source. TargetID, PersonID and GroupID can be used to restrict which clients receive an event. Their use is discussed in Section 4.4 on event delivery and ordering. That section also discusses the SequenceID, NumAccesses, and TimeToLive fields, which are used to control event ordering. The final field, the TimeStamp field, is provided primarily as a way to do logging of sessions using the Event Heap. In addition, the field can be used to provide synchronization between applications interacting through the Event Heap.

4.4 Event Delivery and Ordering

As mentioned earlier, the mechanism of use for Event Heap participants is to either create an event and its associated fields and post it to the Event Heap, or to create a template event with fields filled in with desired values and request events matching the template. Calls are provided in the API for both of these actions.

Event delivery is controlled by source applications setting their SourceID, TargetID, GroupID and PersonID to specific values, and having clients match on specific values for those same fields. A client can receive from only specific sources by setting the SourceID field in their template event to values corresponding to the sources from which they desire to receive events. In practice most event routing is done by having the source application set the TargetID to that of the client application they wish to notify. A variety of communication modes are possible depending on how the fields are set, as shown in Table 2.

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

Table 2- Event Heap Communication Types

Ordering of events is controlled with the SequenceID, NumAccesses and, peripherally, the TimeToLive field. Unlike in TSpaces, which has destructive and non-destructive gets, there is only one Get method for the Event Heap; the NumAccesses field determines whether the Get is destructive by specifying how many times the event may be accessed before it is removed. NumAccesses is decremented for each access by an application and the event is removed when this count reaches zero. Typically its initial value is either infinity for a message of broad interest or 1 for a message sent to one recipient or requesting an action that should be done only once.

Event ordering is fairly straightforward in the case of NumAccesses set to one. In this case applications that are waiting for these events get them from the Event Heap in the order which they arrive and they are immediately removed. Handling infinite NumAccesses is more tricky: the semantics of the TSpaces API's are such that a repeated match on an event template would return the same event over and over again.

We get around this by maintaining a special token tuple per event type. The token has a single field containing the current sequence number for events of that type. Applications posting an event atomically take this tuple, copy the sequence number to their SequenceID field, increment the sequence number, and replace the tuple. The atomicity is provided by the TSpace transaction mechanism, and is necessary to ensure that clients don't die while holding the unique token tuple. Clients begin by matching on SequenceID fields greater than one, and then increment events of interest every time they get a new event. This maintains a global ordering per event type as well as proper ordering at each of the clients.

One potential problem with this technique is that clients that join an Event Heap late may receive many old, non-applicable events before getting to ones currently relevant. This is mitigated by the TimeToLive field which specifies how long (in time units, not number of accesses) the event is relevant. After this time period it is removed by the Event Heap's garbage collection mechanism.

In practice, there are two types of applications that tend to run in an interactive workspace: active participants, and monitoring programs. The latter are applications which monitor events sent through the Event Heap to keep track of the room, log actions, or display activity to the user. Since these applications don't actively respond to events, a snooping version of the Get operation is available that returns matching events without changing the NumAccesses field.

4.5 Naming Considerations

One important consideration in a flexible system like the Event Heap is how to manage the namespace for events, client and server applications. Events themselves are uniquely determined by the EventType field. Currently program authors may choose the value for their event type at their discretion, and authors of applications that want to interact must agree on the event type and its field ordering.

This may seem to be a recipe for chaos, but the nature of the Event Heap makes the situation much better than it may first seem. For example, a collision between events with the same name is unlikely to have deleterious effects. Since events are self describing, applications query and set field values using their string names, not their indices. So, even if two applications were to choose the same event type, as long as their field names were different, receivers would not be able to extract values from fields of the conflicting event type. Of course, it is the receiver's responsibility to handle this case, but receivers must incorporate error handling code anyway.

Another benefit of self-describing events is that a carefully programmed application can learn about new event types just by watching the Event Heap. A user could then be prompted to map the fields of that event onto appropriate behavior within the application.

The namespace for client and servers is also currently controlled by application writers, who choose meanings for values sent in SourceID and TargetID fields. Since the Event Heap is partitioned by event type, this has turned out not to be a problem.

While our current technique for managing the namespace has worked so far, as we scale to more complicated interactions we will probably need a more formal allocation and management scheme. This is an important issue that we intend to address in more detail in the future.

4.6 Paths to the Event Heap

One of our design goals was to support a heterogeneous collection of machines and legacy applications. To do so we have implemented a variety of paths through which applications can communicate with the Event Heap. Some of these are designed primarily to allow new applications to be created, and others are more useful to allow legacy systems to interact with the Event Heap. These paths are shown in Figure 4.

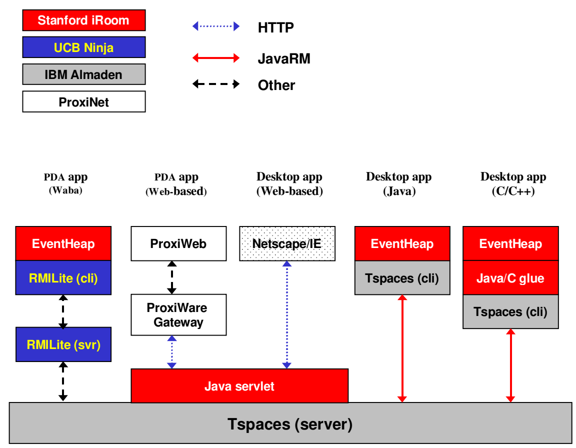


Figure 4 - Application Paths into the Event Heap

The typical way of creating an Event Heap enabled application is to use either the Java or C/C++ API libraries. TSpaces itself is written in Java, and both it and the Java version of the API are portable to any platform supporting JDK 1.1.8 or higher. The JNI interface is used to create a virtual machine inside any C or C++ application using the C/C++ version of our API. The embedded JVM runs the underlying TSpaces Java code. The C/C++ version works under both Linux and Win32 platforms.

For PDA's, including both Windows CE and Palm Pilots, application development is supported under Waba [19], which is a sub-set of Java that can be run on PDA's. There are several key restrictions to Waba that prevent TSpaces from running directly under Waba: there is no threading and RMI is not supported. Since the Event Heap relies on TSpaces, we use RMILite from U.C. Berkeley's Ninja project [5] to provide a means of making TSpaces calls. RMI Lite is a stripped down version of RMI that passes calls from Waba on a PDA to a proxy running on a server machine.

We have also created a path for event submission through the web. This is accomplished by means of a Java servlet, which we call an *usher*, that takes HTTP forms submissions and generates an event in the Event Heap based on information encoded therein. Since form submissions can be encoded in a URL, it is possible to create links on web pages that submit arbitrary events into the Event Heap. The servlet is also set up to encode the desired return URL with the form submission. The ability to submit events using web pages makes it very easy to set up control interfaces that are accessible from any machine, whether transient or permanently in the workspace. This mechanism is described in more detail in [4].

Most PDA's do not support native web browsers, so the ProxiWeb [12] browser can be used to view web pages on them. ProxiWeb works by serving all web pages for the PDA through a third party server which reformats the pages for viewing on the small screen format. Using this mechanism and simple web pages, PDA's can be provided with basic access to functions of an interactive workspace.

Using the described paths and software API's, the Event Heap is currently supported on Windows, Linux, Palm OS, and Windows CE.

5 APPLICATIONS

The Event Heap infrastructure system has been in use since October of 1999 in the iRoom. It has provided an easy way of testing application configurations for the room and has proven to be remarkably robust and stable, in part due to the TSpaces code underlying the Event Heap.

More than a dozen Event Heap-aware applications have now been written. Since most of them use the Event Heap in a similar fashion, in this section we will present the applications most relevant to the functionality described in the motivating example in Section 2. These applications use the Event Heap to control the physical environment (hardware in the iRoom), move information between displays, link data views between applications, and redirect input from I/O devices.

- **Projector Controller:** Used to turn on projectors from a tablet computer, and to automatically turn on table display when there was information for it.
- **Multibrowsing:** Used to provide links from meeting schedule document to pulling up map and 3d model information on other screens.
- **4D Viewer:** Adjusts 3d model view of construction site based on clicking on a map on the table.
- **PointRight:** A system to redirect pointer and keyboard input between multiple machines. This is discussed in detail in [7].

5.1 Projector Controller

The projectors we use in the iRoom can be controlled through the serial ports of host machines. Our projector controller application is a daemon *(projcontrold)* that runs on machines in the room that are connected to the projectors. They wait for projector control events sent over the Event Heap that have their TargetID. When they get these events they send the appropriate signal over the serial port to the projector. The most common commands are to turn the projector on or off, and to tell the projector to switch to displaying a different screen, such as the signal from the laptop drop. Figure 6 shows how events are sent to *projcontrold*. It also shows event paths for multibrowsing, which will be discussed next.

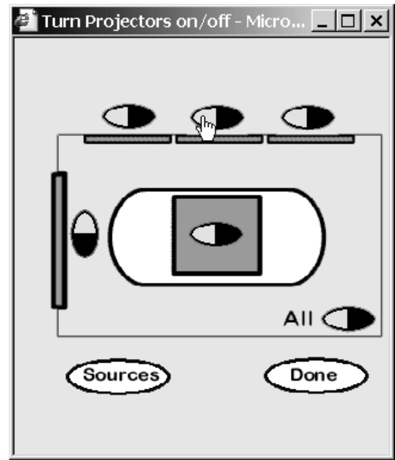


Figure 5 - Web Based Projector Control

Events for the projector control daemons are generated in two ways. The Java based room controller is a custom application that integrates Event Heap code to allow it to submit the projector control events. The second method is using the web path discussed in Section 4.6. Users can click special links on web pages that perform the appropriate form submission for the action specified by the link. One such controller that uses an image map is shown in Figure 5.

5.2 Multibrowsing

Multibrowsing is a system that allows one to call up web pages or other data on a machine by submitting a multibrowse event. It works in a similar fashion to the projector control system. Each machine that is a valid target for multibrowsing runs a multibrowse daemon that waits for events with its TargetID. The events encode a particular command to execute on the target machine. Since the daemon uses Windows shell extensions, URLs are brought up in the default web browser for that machine, and data files, such as Power Point presentations, are brought up in the appropriate application. Executable applications can also be submitted, in which case they are run by the multibrowse daemon. The submission paths for multibrowsing are shown in Figure 6.

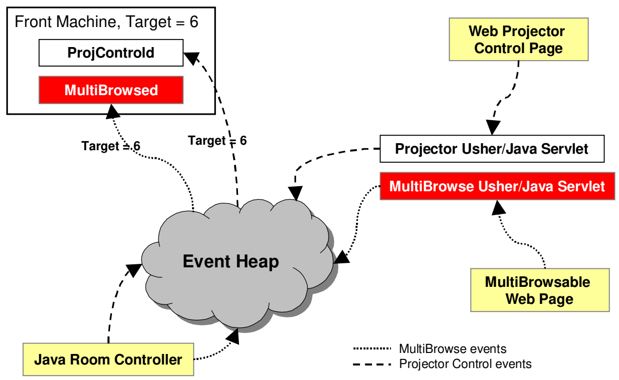


Figure 6 - Event Heap Control Paths

As with projector control, both web pages and custom applications can submit multibrowse events. The same room controller that can change projector settings also allows users to drag and drop information onto an iconic representation of the screens in the room. It then submits a multibrowse event to that machine to load that information. This provides a simple mechanism to bring up the data you want anywhere in the room.

In a running version of the scenario presented in Section 2 a web page with tasks for the meeting was created using Macromedia Flash [11]. The links on the page submit multibrowse commands to load web pages with data on various displays in the room. One of the links also brings up a 3D construction site visualization tool by submitting the command to run the application.

5.3 4D Viewer

The CIFE group at Stanford [8] has created a custom application that shows a 3D model of a construction site as it will appear at any selected point during building, which they call 4D Viewer. Their application writer was able to integrate the Event Heap into their application so that it listened for view change events. They also constructed a web page with an image map of an aerial view of the layout of the construction site; clicking on the image map uses the web-forms event submission path to submit view change events, which are routed to the 4D viewer running on a different display. The work required to enhance their application to use the Event Heap and take advantage of the multiple displays in the iRoom took just a few days.

5.4 Other Event Heap Applications

To give some idea of the scope of what has been done with the event heap, here are some other examples of applications we have running in the iRoom:

- **Light Control via X10:** allows selected lights in the room to be turned on and off using event submission.
- **SmartPPT:** Allows a PowerPoint presentation to be authored for multiple screens and then displayed across the 4 vertically oriented screens in the room.
- **PDA PowerPoint:** Allows a PowerPoint presentation to be displayed and annotated on a PDA, and allows audience members to exchange comments and questions via the Event Heap during a PowerPoint presentation.
- **Collaborative Film Editing:** A team of artist and director can quickly scan in images, display them in an image sorter on one screen, and then compose them into an animated storyboard in Adobe Premiere on two of the other screens.
- **Sound Server:** Allows sounds to be spatially located on a surround sound system and controlled through the Event Heap.
- **PointRight:** Allows pointer/keyboard connected to any machine to be used to control machines displaying to the screens in the room.

Many of the above were implemented as class projects for a course taught on Interactive Workspaces in the Fall of 1999. They were done over about a one-month period by teams of three to five students, which gives some idea of the relative ease with which applications can be developed using the Event Heap.

6 DISCUSSION

Ease of implementation, the ability to leverage many different devices, and the ability to isolate failures have proven to be the Event Heap's biggest strengths in our experiences over the past six months.

6.1 Ease of Implementing Interactive Workspace Applications

By designing a simple API and multiple access paths for the Event Heap, we intended for it to be relatively straightforward to get applications running in the iRoom. This has generally proven to be true.

As mentioned earlier, the six student groups in the class were able to create programs with relatively complex behavior in about one month. The software was also simple enough that only one or two bugs showed up in our initial implementation of the Event Heap during that period. The simple interface also allowed legacy applications to be hooked into the Event Heap with a minimal amount of coding. This allowed the student groups to focus on the inter-application communication structure rather than on the details of the Event Heap framework.

The web interface to the Event Heap has also been very important, making it easy to create web pages that allow integrated display across the screens in the room. Using Macromedia Flash we have been able to mock up even more complex behavior by hiding the URLs in menus and animated buttons. Although the web interface offers more limited functionality than writing applications directly, it provides a path for non-programmers to easily exploit the iRoom's multiple displays and controllable environment.

The one access path that has not been as easy to use is development on PDA's. While the mechanism provided has worked, the development environments available for PDA's and the relative lack of maturity of PDA wireless communication have made implementation of applications on the PDA's somewhat frustrating.

6.2 Support Across a Variety of Devices

Another success for us has been our ability to integrate a heterogeneous collection of devices and machines. The display machines in the room are running various flavors of Windows, including Windows 98, Windows NT and Windows 2000. A Linux server runs the TSpace for the Event Heap and the PointRight software for mouse control.

Portable devices have also been used. The PDA PowerPoint application allows Windows CE users to view slides while in the room. The web interfaces have also allowed both Palms and Windows CE devices to control the environment in the room, and cause web pages to be brought up on the various screens in the room.

6.3 Failure Isolation

For the last several months, all of our group meetings have been held in the iRoom, and we have given numerous demos there. Overall the system has proven to be very reliable. Even when interactive workspace software on a particular machine periodically crashed, the rest of the room continued to function correctly, and when we have rebooted the machine or restarted our software, that machine immediately returned to being a functioning part of the room. This property made demos much less stressful.

Our one failed demo was due to running the TSpace server under an unstable Java virtual machine under Linux. This raises the issue of the TSpace server being a single point of failure for our system. Although TSpaces is a logical single point of failure, nothing about the TSpaces architecture prevents it from being implemented on multiple machines to exploit some redundancy, and we are aware that continuing work at IBM is exploring this avenue.

7 RELATED WORK

Although a few research environments have been constructed that fit the description of an interactive workspace [2][3][9][12][18], they have been designed to investigate specific topics such as computer vision, distributed object systems, and agent interaction, so they have not focused on developing a robust general-purpose infrastructure for interactive workspaces. The specific requirements for such an infrastructure have not, therefore, been very well defined.

The iLand environment built at GMD-IPSI in Darmstadt [18] is physically similar to our interactive workspace. Their BEACH software platform is built in Smalltalk, based on an object-oriented framework called COAST for synchronizing multiple simultaneous access to objects. This enables them to develop sophisticated groupware environments that depend on object synchronization, but it does not support the use of standard UNIX or Windows applications as regular components of the environment. In contrast, it is a design goal of the Event Heap to be able to easily integrate existing applications, including web browsing and desktop applications augmented with collaborative behaviors (e.g. the SmartPPT application mentioned previously).

Jini [20] provides a rendezvous mechanism for Java-based entities to communicate. The Event Heap works at a higher layer than Jini, and is more directly comparable to JavaSpaces [15]. Like TSpaces, JavaSpaces implements a tuple space in the Java environment. We chose to use TSpaces instead to leverage some of its querying semantics, but we believe the Event Heap could also be easily built on top of the JavaSpaces system.

In [6], 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 broadcast/subscribe meshes well with the function of the Event Heap, although we use a more flexible event typing scheme which keeps things less tightly bound.

MacIntyre and Feiner present the COTERIE system in [6]. Their system provides language level support for distributed virtual environments. It provides applications with the ability to create shared objects that are accessible across machines.

8 FUTURE WORK

While the Event Heap is already useful as currently implemented, there are several things we hope to improve. The first is to add event streams as part of the API. Event streams would allow an application to sign up to generate a sequence of events of a certain type, or subscribe to all events matching a certain template event. This provides syntactic sugar for applications to make coding simpler, and also would allow us to open direct socket connections "under the covers" between sources and receivers of low latency events. We already use a limited subset of this functionality for low latency events, since the performance of TSpaces is not fast enough to stay below the perception threshold (although we are aware that the TSpaces team is working on this problem). This fast path would be automatically used whenever an application specifies an event with TimeToLive of zero.

We also plan to construct a set of flexible Event Heap managers. Currently routing of events between applications is controlled by how the application writers choose to use the mandatory fields, and consequently tends to be fairly static and hard-wired to the specific layout of the iRoom. We plan to extend the Event Heap so that it will report the types of events applications use, and allow the TargetID, GroupID, and PersonID fields to be set by other applications managing the room. These managers will be able to reconfigure applications to work in different workspaces, and control which applications interact with on another. They will serve a similar purpose in an Interactive Workspace to window managers on desktop systems.

Workspace managers will also make it easier to set up event intermediation. For example, applications that strokes for entry can be connected to a gesture recognizer, which in turn subscribes to mouse events. We have begun to explore this issue and some initial work is described in [13].

Further consideration is also needed in the area of namespaces. Part of adding support for managers will be the ability to give applications specific names that are well understood by other applications in the interactive workspace. To manage the configuration of the workspace we are currently working on a database for the Interactive Workspaces infrastructure. Applications will be able to use the database to determine configuration and names, and will also be able to register to receive an event when the database is updated.

In the long term we hope to extend the Event Heap to be a framework to support ubiquitous computing in general. This will include the ability to bridge Event Heaps in different interactive workspaces so that people can interact remotely. We hope this will enable more sophisticated distance learning, remote collaboration and tele-conferencing with a simpler implementation route.

9 CONCLUSION

In this paper we have presented the Event Heap, an enabling infrastructure for interactive workspaces. The Event Heap avoids the problems caused by using an event queue in the distributed environment found in an interactive workspace by decoupling communication between applications—instead of direct connections, events are exchanged through a bulletin board mechanism. Our event specification is also flexible, so applications need not be recompiled when structures change as long as fields are not removed from the specification. Finally, our system has been in day to day use for over six months now with few major crashes or problems, demonstrating that it is workable in real world situations. We hope that the cross-platform, legacy-application-friendly nature of the Event Heap will encourage others to use it in their ubiquitous computing work.

Acknowledgments

The Interactive Workspaces project is the result of efforts by too many students to name, both in our research group and in the Interactive Workspaces course. Bryn Forbes, Greg Hutchins, Emre Kiciman, Brian Lee, Shankar Ponnekanti and Rito Trevino helped with the implementation of the Event Heap and various pieces of the infrastructure described in this paper. Kathleen Liston, Meenakshy Chakravorty and several others from the CIFE project implemented the Civil Engineering scenario. Toby Lehman and the TSpaces team at IBM Almaden receive our grateful thanks for putting together a stable base on which to build the Event Heap and supporting our efforts to use it. Finally, we give special thanks to Susan Shepard for keeping the iRoom functional, and to John Gerth for additional administrative support. 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] Cisnero, N., and D.Gelerntner, Applications Experience with Linda. *Proc ACM Symposium on Parallel Programming* ACM, 1988, New Haven CT

[2] Coen, M., Building Brains for Rooms: Designing Distributed Software Agents. In Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence. (IAAI97). Providence, R.I. 1997.

[3] Darrell, T., Maes, P., Blumberg, B., and Pentland, A., ``A Novel Environment for Situated Vision and Behavior'', in Proc. IEEE Workshop on Visual Behaviors, IEEE Computer Society Press, Seattle, 1994.

[4] Fox, A., Johanson, B., Hanrahan, P., Winograd, T., "PDA's in Interactive Workspaces," Computer Graphics and Animation, May, 2000.

[5] Gribble, Steven D., Welsh, M., Brewer, E., and Culler, D. The MultiSpace: an Evolutionary Platform for Infrastructural Services. Proceedings of the 1999 Usenix Annual Technical Conference, Monterey, CA, June 1999. Also available at http://ninja.cs.berkeley.edu/pubs/pubs.html.

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

[7] Johanson, B., Hutchins, G., Winograd, T., "PointRight: A System for Pointer/Keyboard Redirection Between Multiple Displays and Machines", submitted to UIST'2000, San Diego, CA, USA, 2000.

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

[9] Lucente, Mark, Gert-Jan Zwart, and Andrew George, Visualization Space: A Testbed for Deviceless Multimodal User Interface, AAAI Spring Symposium Series, March 23-25, 1998, Stanford University, p. 84.

[10] Blair MacIntyre and Steven Feiner. Language-level support for exploratory programming of distributed virtual environments *Proceedings of the ACM Symposium on User Interface Software and Technology* November 6 - 8, 1996, Seattle, WA USA Page 83 (http://www.acm.org/pubs/citations/proceedings/uist/237091/p83-macintyre/).

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

[12] MIT Media Lab, Smart Rooms Project, http://vismod.www.media.mit.edu/vismod/demos/smartroom/

[13] Michelle Munson and Armando Fox, "Dynamic Control in Tuple Spaces for Sustainable Evolution in Pervasive Computing Applications". To appear in IBM Sys. J., special issue on Pervasive Computing

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

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

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

[17] Spreitzer, M. and Begel, A. More Flexible Data Types. In Proc. Eighth IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET-ICE 99), 1999.

[18] 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.

[19] WabaSoft Inc. Waba virtual machine and documentation. See http://www.wabasoft.com.

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

[21] Terry Winograd, Towards a Human-Centered Interaction Architecture, to appear in J. Carroll, ed., Human-Computer Interaction in the New Millennium, 2000, Addison-Wesley, in press. Available as working paper: http://graphics.stanford.EDU/projects/iwork/papers/humcent/

[22] 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.